This microfiche was produced according to ANSI/AIIM Standards and meets the quality specifications contained therein. A poor blowback image is the result of the characteristics of the original document.

July 1993

Report No. STAN-CS-93-1482

Also numbered KSL 93-58

Thesis

PB96-149612

Irrelevance Reasoning in Knowledge Based Systems

by

Alon Y. Levy

Department of Computer Science

Stanford University

Stanford, California 94305

IRRELEVANCE REASONING IN KNOWLEDGE BASED SYSTEMS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Alon Yitzchak Levy
July 1993

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Richard E. Fikes
(Principal Co-Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Edward A. Feigenbaum
(Principal Co-Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Patrick J. Hayes

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Nils J. Nilsson

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Doctor of Philosophy.

Yehoshua C. Sagiv

Approved for the University Committee on Graduate Studies:

Abstract

Speeding up inferences made from large knowledge bases is a key to scaling up knowledge based systems. To do so, a system must have the ability to automatically identify and ignore information that is irrelevant to a specific task. Identifying irrelevant knowledge is also key to enabling reasoning in environments in which several systems (and their respective knowledge bases) interoperate. This dissertation considers the problem of reasoning about irrelevance of knowledge in a principled and efficient manner. Specifically, it is concerned with two key problems: (1) developing algorithms for automatically deciding what parts of a knowledge base are irrelevant to a query and (2) the utility of relevance reasoning.

As a basis for addressing these problems, we present a formal framework for analyzing irrelevance. The framework includes a space of possible definitions of irrelevance, based on a proof theoretic analysis of the notion. Within the space of definitions, we identify the class of strong irrelevance claims, that has two desirable properties. Strong irrelevance claims can be efficiently derived automatically and are guaranteed to lead to savings in inference.

The dissertation describes a novel tool, the *query-tree*, for reasoning about irrelevance. Based on the query-tree, we develop several algorithms for deciding what formulas are irrelevant to a query. These algorithms dramatically speed up inference, especially when the knowledge base includes a large data base of ground facts. The query-tree has been investigated primarily for Horn rule knowledge bases with interpretable constraints (e.g., order and sort constraints), and several more expressive extensions. For certain cases, the algorithms are shown to be complete, in that they detect *all* the irrelevant formulas. An important aspect of the query-tree is that it can be built by examining only a small part of the knowledge base (e.g., only the rules), and therefore, can be built efficiently. The query-tree is also used to derive the consequences of irrelevance knowledge given by a user. The dissertation presents an empirical analysis of the algorithms when doing backward chaining on Horn rules, showing that in practice, significant savings (often orders of magnitude) are obtained by relevance reasoning.

Our general framework sheds new light on the problem of detecting independence of queries from updates. We present new results that significantly extend previous

work in this area. The framework also provides a setting in which to investigate the connection between the notion of irrelevance and the creation of abstractions. We propose a new approach to research on reasoning with abstractions, in which we investigate the properties of an abstraction by considering the irrelevance claims on which it is based. We demonstrate the potential of the approach for the cases of abstraction of predicates and projection of predicate arguments.

Finally, we describe an application of relevance reasoning to the domain of modeling physical devices. We consider the task of selecting a model for a device and a query by composing *model-fragments*, each describing single phenomena in the physical world at different levels of abstraction and approximation. We present a novel model-composition algorithm based on irrelevance that composes a model with appropriate abstractions and perspectives for answering the query.

Acknowledgements

I would like to thank my adviser Richard Fikes for his support and guidance throughout the work on this dissertation. Through many long discussions, Richard provided me many insightful technical suggestions and helped me clarify my often confused thoughts. He always helped me in focusing my work to the problems that would ultimately lead to important and useful results. He taught me the important lessons of how to validate scientific work and how to present it. Last but not least, my interactions with Richard were always fun, full of good laughs and discussions ranging from good movies to boat rides in Thailand.

I would like to thank my adviser Ed Feigenbaum for his advice and support. Ed took me as a student when all I had was a vague formalism, and gave the freedom to pursue my interests until I finally produced concrete results. His insistence made me realize the importance of real world problems. Through his stories, Ed also instilled in me the desire to travel all around the world.

The story tells that the life of a graduate student contains one major turning point. For me this point occurred at a departmental TGIF party in the spring of 1991, when I met Shuky Sagiv and he insisted that I explain my work to him. From then on, Shuky guided me through the life cycle of scientific work. He taught me how to take abstract ideas and translate them into concrete and defined problems, how to investigate these problems thoroughly until they were fully understood and finally how to write the solutions such that others would be able to understand them. Whether he was in the next room or half the way across the world, Shuky was always there to give me friendly and professional advice about work, the scientific world, and life in general.

I would like to thank my other committee members, Nils Nilsson and Pat Hayes for their advice and the many discussions on my research. I am especially indebted to Nils for his guidance and support during my first years at Stanford and for his concern for my well-being throughout. Pat's thought-provoking questions and perspective were very helpful in crystalizing the ideas underlying my work.

I worked very closely with Yumi Iwasaki. From the day I joined the KSL, Yumi shared and supported my belief that my thesis work is relevant to the domain of modeling physical systems. Until we crystalized our ideas, we spent long hours in

discussions, in which she taught me a lot about qualitative reasoning. I also enjoyed the lessons Yumi taught me about the culture of Japan.

I was very fortunate to have had Pandu Nayak as an officemate and to have him as a friend. Pandu was always ready, at a drop of a hat, to discuss any idea on my mind and tell me immediately what was wrong with it before I even understood it. The great laughs we had in our office were the best of times at Stanford, not to forget our long discussions in Japanese.

With my friend, Ashish Gupta, I shared many of the good times at Stanford. Ashish and I spent many hours discussing various aspects of life and often even technical matters. Ashish was always there to help me fill the gaps in my knowledge of database systems and to cook a good meal when necessary.

I would like to thank other members of the KSL. Bob Engelmore for his support and his comments on my work. Tom Gruber for the many discussions on my work, AI or any other topic that popped up. Jim Rice, without whom the experimental results described in this thesis would not be possible or meaningful. The superb administrative team that keeps the wheels of the KSL rolling: My friend, Grace Smith, who always helped me solve the weirdest of problems, was always fun to talk to, was a constant source of obscure English words and would always laugh at my jokes. Michelle Perrie for all her help and patience especially when fed-ex was involved, and Peche Turner for doing all the right things with the budgets so I didn't have to hear about it.

Several people made life at the KSL fun and provided technical feedback when requested: Diane Chi, Andy Golding, Torsten Heycke, John Mohammed, Janet Murdock, Sam Srinivas, Serdar Uckun, Amy Unruh, Rich Washington and Mike Wolverton.

I would like to thank Hiroshi Motoda for the opportunity to spend three wonderful months with his group at Hitachi Ltd. in Japan, and for his personal support and advice. During my stay in Japan and after, Hiroshi took every opportunity to discuss my work and provide me with useful feedback. My visit to Japan was an extremely joyous experience, largely due to Hiroshi.

At Stanford I had the opportunity to discuss my work with many people. I had exciting arguments with Matt Ginsberg in which I had to vehemently defend my positions. My discussions with Mike Genesereth were extremely useful. I have benefited a lot from discussions with Nir Friedman. Surajit Chaudhuri provided me explanations on various database issues, and in general shared many of my opinions about life. I have also benefited a lot from discussions and comments from the following people: Andy Baker, Adnan Darwiche, Marcia Derr, Don Geddis, Fausto Giunchiglia, Nita Goyal, Adam Grove, Ramanathan Guha, Daphne Koller, Doug Lenat, Inderpal Mumick, Karen Myers, Eunok Paek, Scott Roy, Oded Shmueli, Narinder Singh and Becky Thomas. The following people made life at Stanford fun: Edith Cohen, David

Cyrluk, Raymonde Guindon, Tom Henzinger, Pat Lincoln, Dror Maydan, Tomasz Radzik and Alex Wang.

Financial support for my work by provided by NASA Grant NCC 2-537, a fellowship from Shell Oil Corporation and from Hitachi Ltd.

Finally, I would like to thank my family. My parents, Dvora and Moshe, for their love and support in every endeavor that I have ever taken upon myself, and for encouraging me to pursue my studies. Knowing how much pride they derive from my work and the completion of this thesis provided me with much encouragement throughout my years at Stanford. My sister Sharona for always being a role model for me in her desire to learn and explore.

Last, but by no means least, I would like to thank my wife and best friend Merav. Merav provided me the balance I needed in my life, without which I would have probably gone crazy long ago. Without her love, support, friendship and patience through the hard times, this work would not have been possible. It is to her that I affectionately dedicate this dissertation.

Page intentionally left blank

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The distinguishing characteristic of research in Artificial Intelligence (AI) is that it attempts to automate cognitive tasks that are natural to humans and at which humans are proficient. Prime examples of such research include computer vision, natural language understanding, automatic planning and the formalization of common sense reasoning. In performing cognitive tasks, humans have the natural ability to ignore irrelevant information. We have constant access to very large amounts of information, either in our memory or through external sources. However, when given a specific task, we are able usually to focus on the knowledge that is relevant to that task, thus enabling us to reason in a timely fashion.

In order for machines to be able to reason efficiently in the presence of large amounts of information, they too must be able to ignore irrelevant information. In fact, the inability of current AI systems to ignore irrelevant information is a major obstacle in scaling up such systems. It is well known that the performance of inference engines in AI systems that use declarative representations degrades quickly as the size of the knowledge base increases. Two of the major sources of inefficiency of inference engines are due to this problem:

- In its search for a solution, the inference engine considers many facts in the knowledge base that are irrelevant to the query. Consequently, it spends significant effort pursuing useless solution paths.

- A knowledge base is designed to accommodate a variety of tasks. Therefore, its conceptualization of the domain must be detailed enough for all of them. Consequently, given a specific task the knowledge base is likely to be too complex for it, leading to inefficient reasoning. For example, it may make unnecessary distinctions between objects in the domain or between properties of these objects. In order to achieve efficient performance, an inference engine must be able

to *abstract* automatically the representation by removing irrelevant distinctions in the representation.

Both of these issues will become even more important in the context of future large scale AI systems (e.g., [Fikes *et al.*, 1991; Genesereth, 1992]). Such systems will have access to large amounts of knowledge coming from multiple autonomous sources. The knowledge will overlap in many ways and will be represented in multiple levels of abstraction. Reasoning mechanisms in such systems must be able to decide automatically what knowledge is relevant to a specific task, and what level of abstraction is most adequate.

To illustrate these issues, consider the following simple example knowledge base.

$$flight(X, Y, S_1, E_1, C) \wedge (S \leq S_1) \wedge (E \geq E_1) \Rightarrow path(X, Y, S, E, C)$$
$$bus(X, Y, S_1, E_1, C) \wedge (S \leq S_1) \wedge (E \geq E_1) \Rightarrow path(X, Y, S, E, C)$$
$$path(X, Z, S, E_1, C_1) \wedge path(Z, Y, E_1, E, C_2) \wedge (C_1 + C_2 \leq C) \Rightarrow path(X, Y, S, E, C)$$
$$flight(X, Y, S, E, C) \Rightarrow (C > 70)$$

The atom $flight(X, Y, S, E, C)$ ($bus(X, Y, S, E, C)$) denotes that there is a direct flight (bus) from city X to city Y, departing at time S and arriving at E. The cost of the flight is C dollars. The atom $path(X, Y, S, E, C)$ denotes that there is a path (i.e., sequence of flights and busses) from X to Y, leaving and arriving between S and E and costing at most C dollars. Finally, all flights are known to cost more than \$70. The knowledge base also contains ground atomic facts for the relations *flight* and *bus*.

Suppose we are given the query $path(SF, LA, 8am, 4pm, \$50)$. With respect to this query, the first rule in the knowledge base and all the *flight* ground facts are irrelevant. Ground facts of *bus* that cost more than \$70 or do not run between 8am and 4pm are also irrelevant and can therefore be ignored. Doing so will result in significant savings in answering the query. In contrast, a conventional backward chainer reasoning with this knowledge base will encounter irrelevant facts at various points in its search. In the best scenario, it will immediately realize that a fact is irrelevant (by propagating the constraints) and backtrack. Otherwise, it will continue its search producing a search subspace based on the use of an irrelevant fact, and realize later that the subspace could be eliminated. Even if the backward chainer does realize immediately that a fact that it encounters is irrelevant, there may be many such irrelevant facts, and considering each of these will be very expensive.

Alternatively, suppose we only want to know if there exists some path between two cities using the connections in our knowledge base. In such a case, we can abstract the representation of the domain and modify the rules appropriately. Specifically, the predicates can be reduced from arity 5 to binary (e.g., $flight(X, Y)$ denoting that there is a flight between X and Y). Moreover, the distinction between flights and

busses is also irrelevant, and therefore we can abstract the distinction between *flight* and *bus* (and other travel media we may have). In doing so, we would replace every *flight* (and *bus*) ground fact by a ground fact of a new predicate *directConnection*. Our knowledge base would now be

$directConnection(X, Y) \Rightarrow path(X, Y)$
$path(X, Z) \wedge path(Z, Y) \Rightarrow path(X, Y, S, E, C)$

This knowledge base will yield a much smaller search space and will still enable us to answer the query.

Aside from its use in controlling inference, the need to identify irrelevant knowledge also arises in other contexts in AI:

- **Nonmonotonic reasoning:** In nonmonotonic reasoning, a conclusion drawn from a set of formulas is not guaranteed to hold when additional formulas are considered. Consequently, the inferences made depend in subtle ways on which formulas are considered. A key property that has been the focus of several nonmonotonic formalisms (e.g., [Pearl, 1990; Geffner and Pearl, 1990]) is designing reasoning schemes in which the addition of irrelevant formulas does not change the conclusions. However, the notion of irrelevance has been treated informally thus far in this work.

- **Reasoning by analogy:** Often, properties of one object can be used to conclude properties of another, if there is some analogy between the two objects. However, for the reasoning to be meaningful, the analogy between the objects must be relevant to the property being concluded. Automating such reasoning requires a good understanding of the notion of relevance.

- **Learning:** A drawback of many learning systems is that they produce overly specific descriptions of concepts being learned. This happens when the learned descriptions contain irrelevant information. Using overly specific concept descriptions often degrades the preformance of systems (e.g., EBL). Removing irrelevant information is key to making such concept descriptions useful in problem solving [Etzioni and Minton, 1992].

This dissertation studies the issues involved in reasoning about irrelevance. It presents a general framework and specific methods that enable a system to reason about irrelevance of knowledge to a query. Relevance reasoning is done both by using additional knowledge specified by the user and by automatic methods for analyzing the knowledge base and a specific query. Additional knowledge is specified to the system in the form of meta-level *irrelevance claims* in a language given in the framework.

1.1 Components of the Problem

We break down the problem to the following components:

1. As a basis for stating knowledge about irrelevance and reasoning with it in a principled manner, we must:
   - Formally define the meaning (or meanings) of irrelevance.
   - Identify the different types of irrelevance with which we want to reason.
   - Devise a language for expressing knowledge about irrelevance.
2. In reasoning about irrelevance, we consider two questions:
   - Given a knowledge base and a query, can we decide automatically which facts in the knowledge base are irrelevant to the query (and can we do so efficiently)?
   - How can we derive logical conclusions from meta-level irrelevance claims that are given to the system?
3. Using irrelevance reasoning to control inference:
   - How can we modify inference mechanisms to exploit knowledge about irrelevance?
   - What is the utility of relevance reasoning (in theory and in practice)?

1.2 Overview of the Solutions

We present an overview of the solutions we propose for the questions we address as well as a description of an application of our framework to the problem of selecting models for physical systems.

1.2.1 Analyzing Irrelevance

The notion of irrelevance has been used in many contexts in research in AI and related fields. However, most of the time researchers use the term informally. Formal analyses of irrelevance have been discussed by philosophers as early as [Keynes, 1921], [Carnap, 1950] and [Gärdenfors, 1978]. The main thrust of these analyses was to try to capture our common sense notions of irrelevance by a formal definition. Most of the work focuses on formulating properties of the notion of irrelevance and finding definitions that satisfy the properties. Consequently, the work has not been concerned

with how to *use* irrelevance for speeding up inference or how to design algorithms for detecting irrelevance.

Within AI the notion of irrelevance was investigated in the context of probabilistic reasoning [Pearl, 1988] and used there to control inference in Baysian belief networks. In the context of logical knowledge bases, Subramanian [Subramanian, 1989] investigated several formal definitions of irrelevance. However, the issues of deriving irrelevance claims and the utility of irrelevance reasoning were left largely open.

We want our definitions of irrelevance to make sufficient distinctions to make them useful in developing algorithms for detecting irrelevance. To do so, we analyze irrelevance at the level of the possible derivations (or more generally, solution paths) that a problem solver can pursue in the solution of a goal. In contrast, other analyses have been at the model theoretic level [Gärdenfors, 1978] or the meta-theoretic level [Subramanian, 1989]. Furthermore, we do not purport to provide a *single* best definition of irrelevance. Instead, we provide a space of possible definitions of irrelevance and analyze how the properties of irrelevance change as we move in the space.

We begin by considering the question of defining irrelevance of a single formula to a query. Specifically, if Δ is a knowledge base, q is a query and f is some formula, we will define when f is irrelevant to q with respect to Δ.

The first distinction made in our space is between *weak irrelevance* and *strong irrelevance*. In the former, f will be irrelevant to q if there is *some* derivation of q that does not *use* f. In strong irrelevance, f will be considered irrelevant if it is not used in *any* derivation of q from Δ. Each of these classes can be further refined by considering only a specific set of derivations in the definition. For example, we can define f to be strongly irrelevant to q if it is not used in any *minimal* derivation of q.[1] Furthermore, definitions vary in the way we define what it means for a formula to be *used* in a derivation. For example, we can define f to be used in a derivation D if it appears somewhere in D, or, alternatively, if it is implied by the formulas in D.

Besides irrelevance of formulas, the framework also considers irrelevance of other subjects. For example, we define irrelevance of predicates, objects, refinements of predicates and distinctions between objects. These kinds of irrelevance are later used as justifications for creating abstractions.

The framework has enabled us to make several important distinctions. For example, the class of strong irrelevance claims is shown to have several properties not shared by weak irrelevance. In many cases, it is possible to find *all* strongly irrelevant formulas efficiently. Furthermore, removing strongly irrelevant formulas is shown to speed up inference significantly and is guaranteed never to slow it down. Finally, several instances of strong irrelevance satisfy properties that have been argued to be desirable of a common sense notion of irrelevance in the philosophical literature.

[1] Given some definition of minimality of derivations.

The framework is shown to be general in that it encompasses definitions discussed in the past. These include definitions given by Subramanian [Subramanian, 1989] and definitions given in analysis of databases [Srivastava and Ramakrishnan, 1992] and [Elkan, 1990]. The framework provides important insights into the problem of detecting independence of queries from updates in databases, enabling us to develop new algorithms for solving the independence problem.

1.2.2 Automatically Detecting Irrelevance

A major focus of the thesis is the investigation of the problem of automatically deciding which formulas are irrelevant to a given query. We first address the following question:

- *Given a knowledge base Δ and a query q, which formulas in Δ are irrelevant to q?*

We later use the techniques developed to answer this question in order to solve the problem of deriving logical conclusions from irrelevance claims that are *given* to the system by an external source. We consider the problem for knowledge bases containing Horn rules, and several more expressive extensions.

In general, deciding which formulas are irrelevant to a given query can be more expensive than solving the query itself (without relevance reasoning), especially in large knowledge bases. Furthermore, if the knowledge base changes, the relevance reasoning needs to be repeated. In order for our algorithms to be of practical interest, we must derive irrelevance claims by examining only a small and stable part of the KB, and derive claims that will hold independent of any changes that are made to other unexamined parts. In many applications using Horn rule knowledge bases it is the case that the bulk of the KB is ground facts, and the ground facts are much more prone to frequent changes. Often, the ground facts will be stored in some database. Therefore, we address the following question. Suppose a knowledge base consists of a set of rules $\mathcal{P}$ and a set of ground atomic facts D.

- *Given a set of rules $\mathcal{P}$ and a query q, which rules in $\mathcal{P}$ and which sets of ground facts are irrelevant to q for any choice of ground facts D?*

We consider the problem for several cases of strong irrelevance. For weak irrelevance, the problem is in general undecidable even for simple languages (e.g., no function symbols or recursion). Algorithms that provide sufficient conditions for weak irrelevance are discussed in Chapter 5. The following example illustrates the reasoning we perform.

Example 1.1: Consider the following set of rules:

$r_1 : badPoint(X) \wedge path(X, Y) \wedge goodPoint(Y) \Rightarrow goodPath(X, Y).$
$r_2 : link(X, Y) \Rightarrow path(X, Y).$
$r_3 : link(X, Z) \wedge path(Z, Y) \Rightarrow path(X, Y).$
$r_4 : step(X, Y) \Rightarrow link(X, Y).$
$r_5 : bigStep(X, Y) \Rightarrow link(X, Y).$

The predicates *step* and *bigStep* describe single links between points in a space. The predicate *path* denotes the paths that can be constructed by composing single links. The predicate *goodPath* denotes paths that go from bad points to good ones. A knowledge base contains these rules and various ground facts using the predicates *step*, *bigStep*, *goodPoint* and *badPoint*. Furthermore, we are given that all the ground facts that may appear in the knowledge base satisfy the following constraints:

$badPoint(X) \Rightarrow 100 < X < 200.$
$step(X, Y) \Rightarrow X < Y.$
$goodPoint(X) \Rightarrow 150 < X < 170.$
$bigStep(X, Y) \Rightarrow X < 100 \wedge Y > 200.$

Figure 1.1 is a symbolic representation of the possible derivations of facts of the form $goodPath(X, Y)$. By analyzing the structure of the rules and the constraints appearing in them, it can be seen that rule r_5 will not appear in any derivation of the query, and is therefore strongly irrelevant. Similarly, ground facts in the knowledge base of the form $step(X, Y)$ that do not satisfy $100 < X$ and $Y < 170$ are also strongly irrelevant to the query. ∎

The main difficulty in irrelevance-reasoning is that we need to establish properties of all the possible derivations of the query. However, we need to do it without enumerating all the derivations. To do so, we have developed a novel tool, the *query-tree*. The query-tree is a finite AND-OR tree that symbolically encodes all the possible derivations that can be generated for the query from the given set of rules (the query-tree of Example 1.1 is shown in Figure 1.1). In building the query-tree, we need to address two issues. First, a simple minded top down construction of the tree will not terminate if the rules in the knowledge base are recursive. Therefore, we need some principled method to terminate the expansion of the tree. Second, we need to carefully manipulate the interpretable constraints that appear in the rules in order to be able to derive all the irrelevance claims.

The query-tree generation algorithm addresses these issues by attaching a set of *labels* to nodes in the tree. For example, in Figure 1.1 the labels of the nodes describe the constraints that need to hold on instances of that node in valid derivations. We assign labels to nodes in the tree as we expand it, and we only expand a goal-node

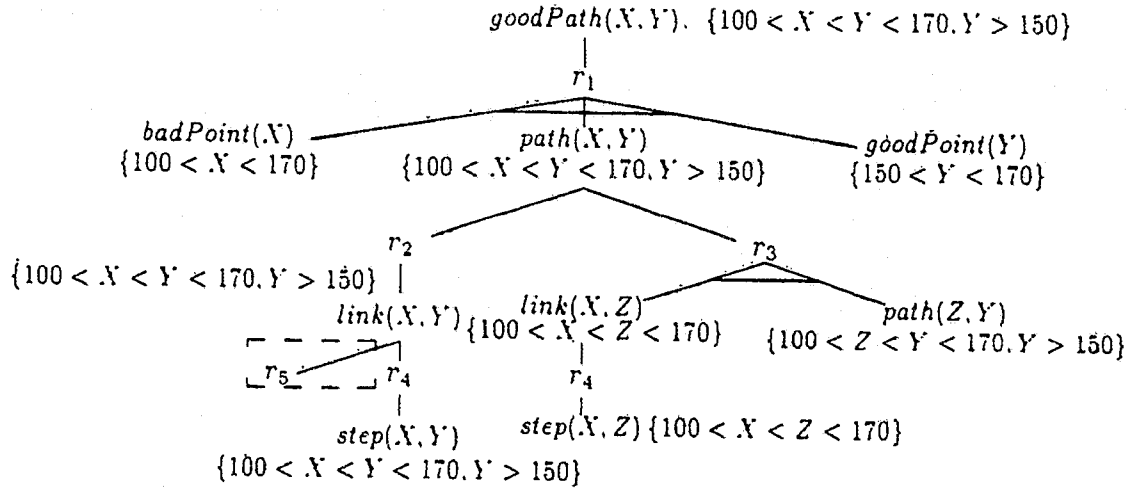

Figure 1.1: An example query-tree

in the tree if there is no other node that is expanded and has an isomorphic label. The labeling scheme of nodes in the tree is chosen to satisfy two constraints. First, the number of possible labels must be finite. This property guarantees that the construction of the tree will terminate. Second, the labeling scheme is chosen such that the resulting tree will encode precisely the set of desired derivations.

If the query-tree encodes precisely the set of derivations of interest, it provides a basis for a sound and complete inference procedure for strong irrelevance. Specifically, a rule is strongly irrelevant to the query if and only if it does not appear somewhere in the tree. For example, in Figure 1.1, rule r_5 does not appear in the tree, and is therefore strongly irrelevant to $goodPath(X, Y)$. A ground fact is strongly irrelevant if and only if it does not match some node in the tree. In Figure 1.1, ground facts of $step(X, Y)$ for which $X < 100$ will not match any node in the query-tree and are therefore strongly irrelevant.

We show that we can devise labeling schemes for nodes in the tree that enable us to encode precisely *all* derivations (and therefore all strong irrelevance claims) for function-free Horn rules that allow a wide class of interpretable constraints (e.g., order constraints, sort constraints). We also discuss a labeling scheme that enables us to encode precisely all the minimal derivations of the query. Finally, we discuss a labeling scheme for encoding precisely all valid derivations of the query when rules may have limited forms of negation in their antecedents.

In some cases (e.g., recursive rules with function symbols) it is not possible to devise an appropriate labeling scheme, and therefore, the query-tree encodes a superset of the valid derivations. In these cases, the query-tree provides only a sound

inference procedure for strong irrelevance. This means that if a rule (or ground fact) does not match some node in the query-tree then it is strongly irrelevant to the query. However, if it does match, that does not necessarily imply that it is not strongly irrelevant. Consequently, using the query-tree enables us to remove only a subset of the strongly irrelevant formulas in the knowledge base.

An important aspect of the query-tree is that it can be built efficiently (and therefore strong irrelevance can be derived efficiently). The size of the query-tree is linear in the number of rules in the knowledge base.[2] Its size depends on the number of different labels we attach to nodes in the tree. This number may be exponential in the arity of the predicates in the KB. However, arities of predicates tend to be very small in practice (e.g., frame systems usually employ only binary predicates). Furthermore, finding examples in which the exponential running time occurs requires careful crafting of the rules. Moreover, since the query-tree is built only based on the rules, it need not be recomputed when the ground facts change. Therefore the cost of building the tree can be amortized over many queries.

The query-tree is related to several graph-like structures discussed in the literature, such as connection graphs [Kowalski, 1975], problem space graphs [Etzioni, 1993], compilation graphs [Bruynooghe *et al.*, 1989] and rule-goal graphs [Ullman, 1989]. The main property distinguishing the query-tree from other structures is the principled treatment of recursion and interpretable constraints. As a result, it is the only structure that computes the tightest constraints on the possible ground facts that appear in derivations, and therefore only the query-tree provides a complete inference procedure for strong irrelevance. Second, the method of building the query-tree is more general than the methods used for building other structures, and therefore we are able to extend it to encode other sets of derivations, (e.g., the set of minimal derivations).

1.2.3 Using Irrelevance to Control Inference

We investigate several methods of using irrelevance reasoning to speed inference:

1. Remove irrelevant formulas: The first method is a simple usage of the query-tree. Given a query (or class of queries) we build its query-tree and decide which formulas are not strongly irrelevant to the query. We then ignore the irrelevant formulas when solving queries of this class, by building a specialized index *only* on the relevant formulas.

2. Ignore irrelevant solutions paths: Aside from encoding only the relevant rules and ground facts, the query-tree also encodes all the *sequences* of rule applications

[2] More precisely, it is linear in the number of rules that are connected to the query through a simple reachability analysis.

that can lead to answers to the query. We show how to modify a backward chainer so that it follows only these sequences.

We describe the results of experiments designed to measure the impact of these savings in practice when performing backward chaining on Horn rules. The experiments show that significant savings are achieved by creating specialized indices, while the cost of building the query-tree and of building the indices is insignificant. Moreover, the results suggest that these methods will scale up to larger knowledge bases and will be especially effective there. For instance, in Example 1.1 when the query-tree deemed 65% of the ground facts in the knowledge base irrelevant to the query, inference was sped up by a factor of 15. When 80% of the ground facts were deemed irrelevant the speedup grew to a factor of 90.

We also discuss how the query-tree can be used to extend more sophisticated query evaluation schemes such as message-passing schemes [Van-Gelder, 1986] and magic set transformations [Ullman, 1989].

3. Detecting irrelevant updates: A frequent operation in persistent knowledge bases is recomputing a query after an update is made to the knowledge base. However, in many cases this computation is wasted because it can be shown that the update will not affect the query, even without actually computing it. We discuss how to detect independence of queries from updates by formulating the problem in terms of relevance reasoning. We show how to use the query-tree and other techniques developed in Chapter 5 to detect independence efficiently.

4. Automatically creating abstractions: As stated earlier, having a more parsimonious representation of the problem domain will lead to more efficient inference. Abstracting a representation to eliminate irrelevant distinctions will result in more parsimonious representations. We show how the creation of abstractions can be posed as a task of relevance reasoning, based on the intuition that a good abstraction is one that removes irrelevant details. We present several algorithms for automatically creating abstractions, based on algorithms for detecting irrelevance.

1.2.4 Irrelevance Reasoning in Automated Modeling

An important domain in which relevance reasoning plays a key role is the domain of modeling physical systems. We apply the framework in this domain to the task of automatically selecting a model for a given system that is appropriate for answering a given query. Briefly, the problem we consider can be formulated as follows. The input consists of three elements:

- Domain theory
- A system description.

- A query about the system.

The domain theory consists of a set of *model fragments* [Falkenhainer and Forbus, 1991]. Each model fragment describes a single phenomenon in the physical world. For example, a model fragment may describe the dependence of the voltage of a battery on its charge level, or it may describe the process of fluid flow through a pipe connecting two containers. The same phenomenon may be described by several model fragments, that differ in the level of detail and the abstractions and approximations made. Each model fragment has a set of *operating conditions* stating when it is applicable. An adequate model of the system is a set of model fragments from the domain theory that have consistent operating conditions.

The system description is a set of facts about the system, including structural specifications and initial values on some of the system parameters. The query is some parameter (or set of parameters), and the answer to the query is a description of the changes of that parameter over time.

The output of the model formulation problem is a set of model fragments from the domain theory that can be used to answer the query about the device. The goal of the model formulation problem is to find the simplest model that can explain coherently the value of the query parameter over time.

Our approach to the problem is based on the observation that several of its aspects can be viewed as instances of relevance reasoning. First, in order to decide which phenomena need to be included in a model of the device, we need to determine which aspects of the domain are relevant to the specific query. This requires that we follow the possible causal influences on the query parameter. Second, in selecting among multiple model fragments representing different ways of modeling a specific phenomenon, we need to reason explicitly about the assumptions being made by each model fragment. We show that many of these assumptions can be stated as irrelevance claims about some aspects of the domain. Finally, the focus of our application is to select a model for *simulation* of the device over time. However, we do not know all the conditions that may arise in the states of the simulation. Therefore, our task is complicated by the fact that we need to select a model based only on partial knowledge about the states that may occur in the simulation. This is analogous to the problem faced by the query-tree of deciding irrelevance by inspecting only a small part of the knowledge base.

We describe a novel model formulation algorithm based on the above observations. The algorithm selects the phenomena that can affect the query by following the possible causal influences on the query parameter. After selecting a phenomenon to include in the model, it chooses among the multiple ways of describing the phenomenon by reasoning explicitly about the different assumptions distinguishing the various descriptions, and choosing the simplest one that does not contradict assumptions made earlier. In making the selections the algorithm also incorporates domain

specific knowledge that may be available from experts. This domain specific knowledge is expressed using the language provided in the framework.

The algorithm has several desirable properties. First, it is guaranteed to select an adequate model for the simulation. Second, given the partial knowledge it has about the possible states of the simulation, it selects the simplest possible model. Finally, the running time of the algorithm is polynomial in the size of the resulting model.

The algorithm has several advantages over previous algorithms [Nayak, 1992a; Falkenhainer and Forbus, 1991; Addanki *et al.*, 1989]. First, it addresses the problem of formulating a model for simulation without creating a complete envisionment of the possible states (as in [Falkenhainer and Forbus, 1991]). Second, the following of possible causal paths by the algorithm frees the user from specifying possible causal interactions explicitly (as in the *component interaction heuristic* [Nayak, 1992a]). This advantage is important since specifying these interactions is a laborious and error-prone task. Finally, unlike the algorithm proposed by Nayak [Nayak, 1992a] that begins with the most complicated model and iteratively simplifies it, our algorithm starts with the simplest model possible and makes it more complex only as required by the modeling assumptions. Consequently, our algorithm is more likely to scale up to larger devices.

1.3 Contributions of the Thesis

The thesis makes the following important contributions:

- It presents a general proof theoretic framework for analyzing irrelevance. The framework crystalizes the issues involved in relevance reasoning; it makes several important distinctions in the analysis of irrelevance, and it generalizes and unifies previous analyses.
- It presents a novel tool, the query-tree, for reasoning about irrelevance. The query-tree is used for:
  - Developing efficient algorithms for detecting strong irrelevance. The algorithms are sound and complete for knowledge bases containing function free Horn rules with a wide class of interpretable constraints. The query-tree can also be designed to be complete for deriving strong irrelevance restricted to minimal derivations and for rules having limited forms of negation in their antecedents. For arbitrary Horn rule knowledge bases, the query-tree provides a sound inference procedure for strong irrelevance.
  - Pushing the tightest constraints possible from a given query to the ground facts in the database. Consequently, a filter can be applied to the ground facts before evaluating the query.

  - Algorithms for deriving logical conclusions from irrelevance claims that are given to the system by an external source.
  - A backward chaining algorithm that is guaranteed not to pursue useless paths.

- It describes experimental results showing that in practice, relevance reasoning leads to significant speedup of inference.
- It describes an application of the framework to the problem of detecting independence of queries from updates, resulting in:
  - New decidable cases for detecting independence.
  - Novel algorithms providing sufficient conditions for independence.
- It describes an application of the framework to the problem of modeling physical systems. Along with providing important insights into that problem, it presents a novel algorithm for automated model selection for simulation based on relevance reasoning.
- It presents a formal connection between relevance reasoning and reasoning with multiple levels of abstraction. The connection enables better analysis of the utility of reasoning with abstractions and the development of algorithms for automatically creating abstractions for a given query.

The main contributions of the thesis are in the fields of knowledge representation and control of reasoning. It is important to note some of the contributions that the thesis makes from the perspective of related fields. Some were outlined in the opening of this chapter, and several are mentioned below.

Deductive databases and logic programming: Much of the work in the thesis can be couched in the terminology of these fields. A major contribution of the query-tree is that it shows how to push constraints from the query to the database. Consequently, a filter can be applied to the database before query evaluation, leading to significant savings. The query-tree and the algorithms discussed in Chapter 3 make significant contributions to the optimization of query evaluation and of logic programs. Specifically, the query-tree can be viewed as a method for partial evaluation of constraint logic programs, extending previous methods in this field.

Knowledge acquisition and knowledge engineering: The framework discussed in the thesis provides a basis for acquiring knowledge about irrelevance, both by providing an expressive language and by indicating where additional knowledge is required. The query-tree can also be used as a tool for knowledge acquisition by

illustrating the connections between pieces of knowledge in a knowledge base and by determining the effects of adding knowledge to the KB.

Reformulation: Subramanian [Subramanian, 1989] first analyzed irrelevance with the goal of automating reformulations. The work presented in this thesis advances Subramanian's analysis and suggests specific methods for discovering irrelevance and creating abstractions. Therefore, it provides a basis for research on automatic reformulation.

1.4 Readers' Guide

The thesis is organized as follows. Chapter 2 discusses the issues arising in analyzing irrelevance, and the proof theoretic framework we propose. It also discusses properties of irrelevance within the framework. Chapter 3 discusses the query-tree. It presents the general method for building a query-tree and describes several of its instances. Chapter 4 discusses the usages of the query-tree in speeding inference. It presents details of experimental results validating the impact of the methods in practice. It also describes algorithms for deriving logical conclusions from given irrelevance claims. Chapter 5 investigates the problem of detecting independence of queries from updates. By translating the problem to reasoning about irrelevance, it provides new algorithms for detecting independence. Chapter 6 makes the formal connection between relevance reasoning and the creation of abstractions, suggesting a new approach to research on reasoning with abstraction. The chapter demonstrates the utility of this approach by considering the examples of predicate abstraction and removal of predicate arguments. Chapter 7 discusses the application of the framework to the domain of modeling physical systems. Chapter 8 concludes with a summary of the contributions, and directions for future research.

The relevant related work is discussed at the end of each chapter. Chapter 8 contains a tabular summary of the references to related work made in the thesis. The thesis is written so that it can be read even by skipping the proofs. The proofs that are included in the chapters are only the most important ones or ones that can add to the understanding of the text. Others appear in Appendix A.

Some of the material covered in the thesis appears in shorter conference length publications. The material in Chapter 2 and Section 4.2 appears in [Levy and Sagiv, 1993a]. Chapter 3 is covered in [Levy and Sagiv, 1992; Levy *et al.*, 1993]. The material of Chapter 5 is presented in [Levy and Sagiv, 1993b]. Finally, the material of Chapter 7 is described in [Levy *et al.*, 1992; Iwasaki and Levy, 1993].

Chapter 2

Analyzing Irrelevance

The notion of irrelevance is used in many contexts in AI research. However, it is typically used informally. Our goal is to state declaratively knowledge about irrelevance and to reason with such knowledge in a principled manner. As a basis for pursuing this goal, this chapter presents a general framework for analyzing formal definitions of irrelevance. The framework is based on a proof theoretic analysis of irrelevance. Section 2.1 begins by describing the motivations for analyzing irrelevance. Section 2.2 describes the issues that arise in the analysis of irrelevance and the possible approaches one can pursue. Section 2.3 describes our framework, consisting of a space of definitions of irrelevance. It also discusses properties of definitions in the space. Finally, Section 2.4 formally presents the problem of automatically deriving irrelevance claims.

2.1 Motivations for Analyzing Irrelevance

The main goal of our analysis of irrelevance is speeding up inference, or, more generally, problem-solving. Irrelevance analysis can be used in speeding up inference in two ways. First, by determining that certain formulas in the knowledge base are irrelevant to a given query, the inference engine can ignore these formulas, and therefore prune its search significantly. Second, by identifying distinctions in the representation that are irrelevant to a specific query, we can create an abstract and more parsimonious representation. We can then translate the knowledge base into this new representation, resulting in more efficient inference.

The analysis of irrelevance is also important in other contexts in AI for purposes other than speeding up inference. In some cases an understanding of irrelevance is needed in order to determine which inferences can be made, as the following examples illustrate.

1. **Non monotonic reasoning:** In non monotonic reasoning, the addition of knowledge can cause previous conclusions to be retracted. Consequently, conclusions drawn in non monotonic reasoning formalisms depend in subtle ways on which knowledge is considered. The following example (from [Pearl, 1990]) illustrates that dependency:

   Example 2.1: Consider a knowledge base containing the following:

   Birds typically have wings.
   Birds typically fly.
   Penguins are birds.
   Penguins don't fly

   Suppose our query is: *Do penguins have wings?* The difficulty in answering the query is that penguins are abnormal with respect to flying, and therefore, may be abnormal in other ways too, such as having wings. However, the fact that a specific bird is a penguin is irrelevant to whether it has wings or not, and therefore we would like to ignore the abnormality of penguins and to conclude that penguins do have wings.[1] As another example, suppose our query is *Can red birds fly?* Here too, we are asking about a property of a subclass of birds, which, as with the subclass of penguins, may be abnormal with respect to flying. However, the fact that a bird is red is irrelevant to its flying ability. ∎

   Designing non monotonic reasoning formalisms that are able to ignore irrelevant information has received a lot of attention recently [Pearl, 1990; Geffner and Pearl, 1990; Bacchus *et al.*, 1993]. However, in that work the notion of irrelevance is either treated informally, or the definitions that are used are very simple minded.

2. **Reasoning by analogy:** In reasoning by analogy we conclude properties of one object from properties of another, based on a possible analogy existing between the two objects. However, for the reasoning to be meaningful, the analogy between the objects must be relevant to the property being concluded. For example, suppose we state the analogy: *Fred is like a fire-engine.* Intuitively, we may use that to conclude that Fred is loud, or that his activity level is high. However, it seems improper to conclude that Fred's color is red, or that his fuel consumption is medium, since these properties are irrelevant to the analogy between Fred and a fire-engine.

[1] It may be argued that having wings *is* relevant to the ability of a bird to fly. However, this is not explicitly stated in the knowledge base. The first statement of this example can also be replaced by any property of birds that is completely unconnected to flying.

The notion of irrelevance also plays an important role in designing algorithms for abductive reasoning [Levesque, 1989] and for belief revision [Gärdenfors, 1988].

2.2 Issues in Analyzing Irrelevance

In this section we discuss several of the issues that arise in an analysis of irrelevance, and provide the motivations underlying our approach. Throughout this chapter we will be concerned with defining *formula-irrelevance*, i.e., given a knowledge base of formulas Δ, a query q and a formula f (not necessarily in Δ), when do we say that f is irrelevant to q with respect to Δ. Irrelevance of other kinds of subjects, such as predicates, objects, predicate refinements and object refinements are considered in Chapter 6.

Two Possible Approaches: Common Sense Formalization vs. Problem solving Analysis

Broadly, we distinguish two possible approaches to analyzing irrelevance. The first approach, which has been pursued by several philosophers ([Keynes, 1921; Carnap, 1950; Gärdenfors, 1978]), is to try to capture our common sense notion of irrelevance with a formal definition. In that approach, we would consider a formal definition of irrelevance and check whether it satisfies properties which we consider natural for our intuitive notion of irrelevance.

The second approach is to analyze the ways in which irrelevance arises in problem-solving. Here too, we would consider various definitions of irrelevance and investigate their properties. However, the properties of interest will be those that are informative in designing inference methods that utilize irrelevance. To illustrate this point, consider the following example.

Example 2.2: Suppose we have the following knowledge base Δ_0:

$r_1 : attendClass(X, Y) \Rightarrow pass(X, Y)$.
$r_2 : passExam(X, Y) \Rightarrow pass(X, Y)$.
$r_3 : pass(X, Y) \wedge tookGradCourse(X) \Rightarrow canTA(X, Y)$.
$r_4 : pass(X, Y) \wedge (Y \geq 200) \Rightarrow tookGradCourse(X)$.
$g_1 : attendClass(Fred, 101)$.
$g_2 : passExam(Fred, 101)$.
$g_3 : passExam(Fred, 161)$.
$g_4 : passExam(Fred, 123)$.
$g_5 : passExam(Fred, 202)$.

And let our query be $q : canTA(Fred, 101)$.

Each of the ground atoms g_1–g_4 can be considered irrelevant to the query in isolation, because for each of them, the query can be derived without using them. However, there are differences between these irrelevance claims. For example, g_3 and g_4 will not be part of any derivation of the query, and therefore can be both ignored. Even though the query can be derived without g_1 or g_2, one of them is always needed, and therefore, we can not remove both. The ground atom $canTA(Fred, 202)$ can also be considered irrelevant to the query, but in a somewhat weaker sense. Although it is never part of any derivation of the query, it is always entailed by the formulas used in the derivation of the query.

Consider the query $canTA(Fred, 202)$. The atom $passExam(Fred, 210)$ can be part of a derivation of this query (if it were in the KB), but such a derivation would not be *minimal*, in the sense that the set of ground atoms that it uses from the KB is not minimal (i.e., using g_5 is enough for entailing the query). Finally, the rule

$$passExam(X, Y) \wedge (Y \geq 200) \Rightarrow canTA(X, Y)$$

could also be considered irrelevant, since the query can be derived without it. However, for some inference mechanisms, it may be the case the this rule will speed inference, since the query can be derived using one rule application instead of two. ∎

The analysis presented in this dissertation is based on the ways in which irrelevance arises in problem solving, for two reasons:

1. Our prime concern is speeding up inference, and therefore, we desire that our analysis provide the distinctions necessary to exploit irrelevance in inference methods, such as those illustrated in the example.

2. Moreover, even if we could agree on a *single* best formalization of our common sense notion of irrelevance, it will have many different manifestations in inference depending on the form of the knowledge base and the inference method. It is therefore important to distinguish these manifestations in order to develop methods for speeding up inference.

Clearly, these two approaches to analyzing irrelevance are not independent of each other. On the one hand, the analysis of irrelevance that we consider is inspired by our common sense notion of the concept, and the definitions we examine mirror it in various ways. We will also see that the distinctions made in our analysis correspond to properties of the common sense notion of irrelevance. On the other hand, given a formalization of the common sense notion of irrelevance, analyzing it in our framework will provide a way of using it for speeding up inference. However, it should be emphasized that the approach we have taken is intended to be evaluated on its usefulness for speeding up inference, not on how well it captures intuitive notions of irrelevance.

Irrelevance With Respect to Given Evidence

Much of the work on formalizing irrelevance (including the work in the philosophy literature) has focussed on the following question:

- Given a set of *evidence* $\mathcal{E}$ and query q, which formulas are irrelevant to q with respect to the evidence?

In our analysis, the knowledge base acts as the evidence, and therefore the question we address is the following:

- Which parts of a given knowledge base Δ are irrelevant to q?

The difference between the two questions is that in the first, the set of evidence formulas is given special treatment by being given priority over the other formulas. In the second, the KB Δ acts as our evidence; however, our goal is to find which parts of the evidence are irrelevant to the query. It may seem that the second question can be considered an instance of the first by equating Δ and $\mathcal{E}$.[2] However, several assumptions made in addressing the first question (e.g., [Gärdenfors, 1978]) make it impractical to use the solutions for the second question. For example, one assumption is that any formula $f \in \mathcal{E}$ will be considered irrelevant to the query (since it is already known and does not change the state of affairs). Another property considered is independence of the form of the evidence, i.e., if $\mathcal{E}_1$ is equivalent to $\mathcal{E}_2$, then the formula f is irrelevant to q w.r.t. $\mathcal{E}_1$ if and only if it is irrelevant to q w.r.t. $\mathcal{E}_2$. In Section 2.3.4 we will see that our framework is general enough to accommodate definitions of irrelevance that give special treatment to a subset of evidence formulas.

Irrelevance as Belief Revision

Intuitively, a formula f is irrelevant to a query q with respect to a KB Δ if f can be removed from Δ and q will still be derivable. This intuition can be generalized by relating irrelevance to the notion of belief revision. Specifically, a formula f is irrelevant to a query q w.r.t. a KB Δ, if the result of revising Δ so that it does not entail f does not affect its entailment of q. Formally, let $\circ$ be a belief revision operator. Given a knowledge base Δ and a formula ϕ, $\Delta \circ \phi$ returns a (consistent!) revision of Δ that entails ϕ. Using this operator, we can define irrelevance as follows:[3]

$$Irrelevant(f, q, \Delta) \iff ((\Delta \circ f) \models q \iff (\Delta \circ \neg f) \models q)$$

[2] Another way to relate these two questions is by considering the set of evidence empty. However, in such cases, the proposed solutions to the first question trivialize.

[3] Note that if Δ is consistent, then either $(\Delta \circ f)$ or $(\Delta \circ \neg f)$ will be simply $\Delta \cup f$.

Many definitions have been proposed in the literature for a belief revision operator (e.g., [Dalal, 1988; Winslett, 1990; Fagin *et al.*, 1983; Nebel, 1989; Alchourron *et al.*, 1985]). There is little agreement on a single best definition, and the properties that are widely considered desirable of such an operator (e.g., the AGM postulates [Alchourron *et al.*, 1985]) give us little information about the resulting properties of irrelevance. In this dissertation we address directly the notion of irrelevance. However, investigating connections between our analysis and belief revision is an interesting area of research.

Our Approach

In order for our analysis of irrelevance to be useful, we desire that it enable us to make sufficient distinctions to answer the following questions:

1. Can we decide automatically which formulas are irrelevant to a given query? Can we do so efficiently?
2. If an irrelevant formula is removed, is inference guaranteed to be more efficient?
3. How can we automatically derive additional irrelevance claims? For example, does irrelevance of a formula imply the irrelevance of a syntactically related formula?

In order to capture the distinctions needed to answer these questions, we present an analysis of irrelevance in terms of the possible *paths* that an inference engine may pursue in the solution of a query. We present a space of possible definitions of irrelevance and investigate the properties of various definitions in the space. In our discussion, we focus on inference mechanisms that attempt to construct derivations of answers to the query, and therefore, paths are actually the possible derivations that the inference mechanism may consider in its search. However, the framework is general and can accommodate other types of problem solving methods. An example of other methods will be discussed in Chapter 7 when we consider reasoning about physical systems. As examples of definitions in our space, we may consider f to be irrelevant if there is *some* derivation of q that does not use f, or if f is not used in *any* derivation of q, or if f is not used in any *minimal* derivation of q. The next section presents the space of definitions of irrelevance.

2.3 A Space of Definitions

2.3.1 Preliminaries

In our discussion we assume the theory of the domain is represented by a knowledge base of closed formulas Δ, in first order predicate calculus. We assume that the

inference mechanism employs a set of sound inference rules $\mathcal{S}$. A derivation D of a closed formula ψ from Δ is a sequence of formulas, $\alpha_1, \ldots, \alpha_n$, such that $\alpha_n = \psi$ and for each i $(1 \leq i \leq n)$, either $\alpha_i \in \Delta$, α_i is a logical axiom, or α_i is the result of applying a rule in $\mathcal{S}$ to some elements $\alpha_{i_1}, \ldots, \alpha_{i_l}$ that appear prior to α_i. The formulas $\alpha_{i_1}, \ldots, \alpha_{i_l}$ are said to be *immediate subgoals* of α_i. The set of formulas in D that do not have any subgoal is called the *base* of the derivation, denoted by $Base(D)$. The set $Base(D)$ represents a "support set" for ψ in the knowledge base. We consider only derivations in which every α_i is a subgoal of ψ (not necessarily an immediate subgoal).

A *query* is represented by a formula ψ. If ψ is a closed formula (i.e., has no free variables), then the answer is `true` if the inference mechanism can find some derivation of ψ from Δ, and `false` otherwise.[4] If ψ contains free variables, the answer is the set of assignments for the free variables, such that the resulting closed formulas are derivable from Δ;[5] in this case, a derivation is a set containing a single derivation for each answer. A query may have several derivations from a given knowledge base, and we denote the set of those derivations by $\mathcal{D}_\psi(\Delta)$ (note that if ψ has free variables, then $\mathcal{D}_\psi(\Delta)$ is a set of sets of derivations).

Our goal is to define the meaning of an *irrelevance-claim* stating that a formula ϕ is irrelevant to a query ψ with respect to a knowledge base Δ. The formula ϕ is called the *subject* of the irrelevance claim.

2.3.2 The Axes

As stated, we describe a space of possible definitions of irrelevance. Definitions in the space vary along two axes. In the first axis we consider different ways of defining *derivation irrelevance*, i.e., irrelevance of a subject ϕ with respect to a *single* derivation D of the query ψ. Derivation irrelevance is given by defining a binary predicate $DI(\phi, D)$. The following are a few examples of how DI can be defined:

- $DI_1(\phi, D)$ iff $\phi \notin Base(D)$.
- $DI_2(\phi, D)$ iff $\phi \notin D$.
- $DI_3(\phi, D)$ iff $Base(D) \not\models \phi$.
- $DI_4(\phi, D)$ iff $Base(D) \not\models \phi, \neg\phi$.

[4] We can return **unknown** if neither ϕ nor $\neg\phi$ are derivable. However that does not affect our discussion.

[5] An alternative definition often considered (e.g., in Prolog) is finding *one* variable binding that satisfies the query formula. However, this distinction does not affect our discussion.

Definition DI_1 requires that ϕ not be in the support set of the derivation D. Definition DI_2 is stronger and requires that ϕ not be anywhere in D. Definition DI_3 is even stronger and requires that ϕ not be a logical consequence of the formulas in $Base(D)$, and DI_4 requires that $\neg\phi$ not be a logical consequence either. The relationship between these definitions of DI can therefore be summarized as follows:

Proposition 2.3: $DI_4(\phi, D) \Rightarrow DI_3(\phi, D) \Rightarrow DI_2(\phi, D) \Rightarrow DI_1(\phi, D)$.

Requiring that DI holds for all possible derivations of the query may be too restrictive. Therefore, in the second axis we consider different subsets of the derivations of the query for which we require DI to hold. Formally, given the possible derivations of ψ from Δ, $\mathcal{D}_\psi(\Delta)$, we consider a subset $\mathcal{D}_0(\Delta)$ of $\mathcal{D}_\psi(\Delta)$, (which may be $\mathcal{D}_\psi(\Delta)$ itself), and require that DI hold for derivations in $\mathcal{D}_0(\Delta)$. For example, we can require DI to hold only for the set of *minimal* derivations. In section 2.3.4 we consider several definitions of minimality for a derivation. As another example, we can consider only the set of derivations bounded by some resource constraint.

Given a choice for DI and $\mathcal{D}_0$, we give two definitions of irrelevance, depending on whether DI is required to hold for *all* derivations in $\mathcal{D}_0(\Delta)$ or for *some* derivation in $\mathcal{D}_0(\Delta)$.[6] Formally, a definition of irrelevance in our space is given as follows:

Definition 2.4: Suppose we are given:

1. a knowledge base Δ,
2. a closed formula ϕ (the subject),
3. a query ψ,
4. a predicate $DI(\tau, D)$ specifying when a formula τ is irrelevant with respect to a derivation D,
5. a subset $\mathcal{D}_0(\Delta)$ of $\mathcal{D}_\psi(\Delta)$.

The formula ϕ is said to be *weakly irrelevant* to ψ with respect to Δ, DI and $\mathcal{D}_0$, denoted by $WI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$, if $DI(\phi, D)$ holds for some $D \in \mathcal{D}_0(\Delta)$.
The formula ϕ is said to be *strongly irrelevant* to ψ with respect to Δ, DI and $\mathcal{D}_0$, denoted by $SI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$, if $DI(\phi, D)$ holds for every $D \in \mathcal{D}_0(\Delta)$.
If $\mathcal{D}_\psi(\Delta)$ is empty (i.e., ψ is not derivable from Δ using $\mathcal{S}$), the formula ϕ is both weakly and strongly irrelevant to ψ. ∎

[6] We can also consider other ways of quantifying over the set $\mathcal{D}_0(\Delta)$, such as requiring that DI holds for some percent of the derivations in $\mathcal{D}_0(\Delta)$. Here we consider only universal and existential quantification.

In our discussion we want to refer to irrelevance of a set of formulas. Formally, we define irrelevance of a set of formulas by extending the definition of DI:

Definition 2.5: If Φ is a set of formulas, $DI(\Phi, D)$ holds if $DI(\phi_i, D)$ holds for every $\phi_i \in \Phi$. ∎

The definitions of strong and weak irrelevance remain unchanged. It will also be useful to state irrelevance claims that hold for a set of knowledge bases. For example, in the context of Horn rule knowledge bases, we will want to know whether a rule is irrelevant with respect to all the knowledge bases that differ only in ground atomic facts. We extend the definitions to sets of knowledge bases as follows:

Definition 2.6: Let Σ be a set of knowledge bases. We say that ϕ is weakly irrelevant to ψ with respect to Σ, denoted by $WI(\phi, \psi, \Sigma, DI, \mathcal{D}_0)$, if ϕ is weakly irrelevant to ψ with respect to every KB in Σ, i.e., if $WI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$ holds for every $\Delta \in \Sigma$. The definition for strong irrelevance is extended likewise. Note that $\mathcal{D}_0$ is actually a function that for every given $\Delta \in \Sigma$ returns a subset of $\mathcal{D}_\psi(\Delta)$. ∎

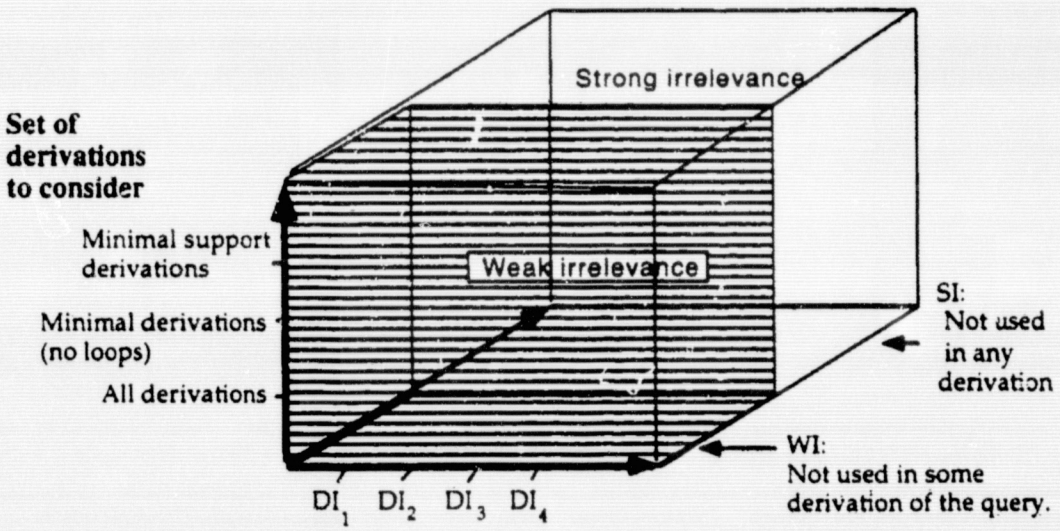

Figure 2.1: A space of definitions of irrelevance. The first axis consists of different definitions of derivation irrelevance. The second axis consists of the set of derivations considered. Weak irrelevance and strong irrelevance differ on the way we quantify DI_i over the derivations chosen in the second axis.

The space of definitions is summarized in Figure 2.1. In Example 2.2 presented earlier, we can see different kinds of irrelevance claims. The atom g_1 (as well as the

atom g_2) is weakly irrelevant to the query $q = canTA(Fred, 101)$, since there is a derivation of q that does not use g_1 (i.e., uses g_2 instead). Consequently, $WI(g_2, q, \Delta_0, DI_2, \mathcal{D}_q)$ holds. The atom g_3 (as well as g_4) is strongly irrelevant to q, because none of the derivations of q use it. Consequently, $SI(g_3, q, \Delta_0, DI_2, \mathcal{D}_q)$ and $SI(\{g_3, g_4\}, q, \Delta_0, DI_2, \mathcal{D}_q)$ hold.

The atom $q_1 = canTA(Fred, 202)$ is strongly irrelevant to q if we consider derivation irrelevance based on DI_2. However, if we consider derivation irrelevance based on DI_3, it is not strongly irrelevant, since the formulas used to derive q can also be used to derive q_1.

Finally, if we consider the set of all derivations of the query q_1, $\mathcal{D}_{q_1}$, then the atom $passExam(Fred, 210)$ is not strongly irrelevant to the query, since it can be used in a derivation of q_1 (to derive $tookGradCourse(Fred)$). However, if we consider only derivations in which $Base(D)$ is minimal (i.e., there is no subset of $Base(D)$ that is enough to derive the query), then $passExam(Fred, 210)$ would not be part of any derivation of q_1, and would therefore be strongly irrelevant to it.

2.3.3 Properties of Definitions In The Space

Several general properties of definitions in the space will be useful in the analysis of specific definitions. The following lemma establishes an ordering on definitions in the space, and will enable us to derive properties of definitions based on properties of other definitions in the space.

Lemma 2.7: *Let DI, DI_i and DI_j be definitions of derivation irrelevance. Let Φ be a set of formulas, ψ be a query and Σ, Σ_1 and Σ_2 be sets of knowledge bases. Finally, let $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2$ be functions that given a KB Δ and a query ψ, return a subset of $\mathcal{D}_\psi(\Delta)$.*

1. *If $DI_i(\tau, D) \Rightarrow DI_j(\tau, D)$ for any set of formulas τ and derivation D, then for any set of formulas Φ, query ψ, set of knowledge bases Σ and set $\mathcal{D}_0$*

$$SI(\Phi, \psi, \Sigma, DI_i, \mathcal{D}_0) \Rightarrow SI(\Phi, \psi, \Sigma, DI_j, \mathcal{D}_0)$$

and

$$WI(\Phi, \psi, \Sigma, DI_i, \mathcal{D}_0) \Rightarrow WI(\Phi, \psi, \Sigma, DI_j, \mathcal{D}_0).$$

2. *If $\mathcal{D}_1(\Delta) \subset \mathcal{D}_2(\Delta)$, for any knowledge base $\Delta \in \Sigma$, then for any set of formulas Φ, query ψ and definition DI:*

$$SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_2) \Rightarrow SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_1)$$

holds. For weak irrelevance, the opposite holds

$$WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_1) \Rightarrow WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_2).$$

3. *For any set* Φ, ψ, DI, Σ *and* $\mathcal{D}_0$

$$SI(\Phi, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow WI(\Phi, \psi, \Sigma, DI, \mathcal{D}_0).$$

4. *If* $\Sigma_1 \subseteq \Sigma_2$ *then*

$$SI(\Phi, \psi, \Sigma_2, DI, \mathcal{D}_0) \Rightarrow SI(\Phi, \psi, \Sigma_1, DI, \mathcal{D}_0)$$

$$WI(\Phi, \psi, \Sigma_2, DI, \mathcal{D}_0) \Rightarrow WI(\Phi, \psi, \Sigma_1, DI, \mathcal{D}_0)$$

Proof: The proofs follow straightforwardly from the definitions. Consider Part 1. Suppose $SI(\Phi, \psi, \Sigma, DI_i, \mathcal{D}_0)$ holds and let $\Delta \in \Sigma$. Therefore, for every derivation $D \in \mathcal{D}_0(\Delta)$, $DI_i(\Phi, D)$ holds and therefore, by the assumption of the lemma, $DI_j(\Phi, D)$ holds. Consequently, $DI_j(\Phi, D)$ holds for every $D \in \mathcal{D}_0(\Delta)$, and so $SI(\Phi, \psi, \Sigma, DI_j, \mathcal{D}_0)$ holds. The proof for WI is similar.

Part 2 about strong irrelevance follows from the observation that if DI holds for all derivations in the set $\mathcal{D}_2(\Delta)$, it will hold also for all derivations in its subset $\mathcal{D}_1(\Delta)$. For weak irrelevance, the claim follows from the observation that if a property holds for some derivation in $\mathcal{D}_1(\Delta)$, it will obviously hold for some derivation in $\mathcal{D}_2(\Delta)$.

Parts 3 and 4 are immediate consequences of the definitions. ∎

An important property of irrelevance claims is whether they are closed under the union of their subjects. This is important when a system needs to determine whether it can use all the irrelevance claims it has, or whether using certain ones will falsify others.

Observation 2.8: Closure under union: Weak irrelevance claims are not closed under the union of their subjects in general. In contrast, for strong irrelevance claims (and combinations of strong and weak irrelevance) we have a sufficient condition for closure that depends only on DI. Specifically, whenever

$$DI(\Phi_1, D) \wedge DI(\Phi_2, D) \Rightarrow DI(\Phi_1 \cup \Phi_2, D)$$

holds for any derivation D and sets Φ_1, Φ_2, then for any choice of $\mathcal{D}_0$ and Σ,

$$SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge SI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow SI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$

and

$$SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge WI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow WI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0).$$

However,

$$WI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge WI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0) \Rightarrow WI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$$

does not hold in general.

The reverse holds for all definitions, i.e., if Φ is irrelevant to ψ and $\Phi_1 \subset \Phi$, then Φ_1 is irrelevant to ψ.

Proof: Suppose $SI(\Phi_1, \psi, \Sigma, DI, \mathcal{D}_0) \wedge SI(\Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$ holds, and let $\Delta \in \Sigma$, and D be a derivation in $\mathcal{D}_0(\Delta)$. Since both $DI(\Phi_1, D)$ and $DI(\Phi_2, D)$ hold, also $DI(\Phi_1 \cup \Phi_2, D)$ holds. Because this holds for any $D \in \mathcal{D}_0(\Delta)$ and $\Delta \in \Sigma$, then $SI(\Phi_1 \cup \Phi_2, \psi, \Sigma, DI, \mathcal{D}_0)$ holds.

The proof for the second claim is similar. We simply consider the derivation D for which $DI(\Phi_2, D)$ holds, and $DI(\Phi_1 \cup \Phi_2, D)$ will hold. The weak irrelevance claims for g_1 and g_2 in Example 2.2 present a counterexample of the third implication. ∎

Observation 2.9: Non-monotonicity: In order to exploit irrelevance claims, it is important to know whether their truth changes when the knowledge base changes. In general, adding new formulas to the knowledge base may cause a formula that was irrelevant to become relevant, or vice versa, a formula that was not irrelevant can become irrelevant. Weak irrelevance claims can change even when the added formulas are logical consequences of the knowledge base. In contrast, strong irrelevance claims are more robust. Definitions of strong irrelevance claims have the property that they do not change when the added knowledge is obtained by reasoning with the original knowledge base.[7] Specifically, if $\Delta \vdash \tau$ and Δ is consistent, then

$$SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}_0) \Rightarrow SI(\Phi, \psi, \Delta \cup \tau, DI_2, \mathcal{D}_0)$$

Hence a strongly irrelevant formula can not become relevant by reasoning on existing knowledge.[8]

Proof: Suppose that $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}_0)$ holds and suppose in contradiction that D is a derivation of ψ from $\Delta \cup \tau$ such that $\phi \in D$ and $\phi \notin \Phi$. We create a derivation D' of ψ from Δ such that $\tau \notin D'$. The only modification to D is to replace every appearance of τ as a leaf in the proof tree by the derivation of τ from Δ. The result is a derivation of ψ from Δ that includes ϕ. Consequently, $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}_0)$ does not hold. ∎

As stated in our original motivations, for most definitions of irrelevance (and in particular all the definitions we consider here), if ϕ is irrelevant to ψ, it can be safely removed from the knowledge base:

Observation 2.10: For any definition of irrelevance that uses a definition of derivation irrelevance DI such that $DI(\phi, D) \Rightarrow DI_1(\phi, D)$, then $\Delta \models \psi$ holds if and only if $(\Delta - \phi) \models \psi$.

[7] This property also seems natural for our common sense notion of irrelevance.

[8] Assuming our inference is monotonic.

Proof: Suppose $WI(\phi, \psi, \Delta, DI, \mathcal{D}_0)$ holds. This means that if $\Delta \vdash \psi$, then $\mathcal{D}_0(\Delta)$ is not empty, and ψ has some derivation D for which $DI(\phi, D)$ holds. The formula ϕ is not a member of $Base(D)$ because $DI_1(\phi, D)$ also holds. Therefore, the derivation D will also be a valid derivation from $\Delta - \phi$. ∎

The utility of removing an irrelevant formula is a more subtle issue. Removing a formula that is only weakly irrelevant may not speed inference. In fact, explanation based learning systems [Minton *et al.*, 1989] do exactly the opposite, they add redundant rules (which, in our framework, would be considered weakly irrelevant). The utility of adding such rules is a subject of ongoing research (e.g. [Minton, 1988; Etzioni, 1990; Greiner and Jurišica, 1992; Etzioni and Minton, 1992]).

For strong irrelevance, savings are guaranteed for many cases. For example, when considering all derivations of the query (i.e., $\mathcal{D}_0 = \mathcal{D}_\psi$), if $SI(\Phi, \psi, \Delta, DI_2, \mathcal{D}_\psi)$ holds, then deriving ψ from $\Delta - \Phi$ costs no more than deriving it from Δ. This property also holds if we consider a set of derivations $\mathcal{D}_0(\Delta)$, such that the inference engine is always guaranteed to find one of the derivations in $\mathcal{D}_0(\Delta)$ before it finds others. Removing strongly-irrelevant formulas yields savings from several sources:

- Removing irrelevant formulas prunes whole branches of the search space.
- Much of the cost of reasoning in a large knowledge base is in doing database lookups. Removing a large number of irrelevant ground facts at the outset will significantly reduce the cost of each lookup operation.
- If updates are made to the KB that concern only irrelevant formulas, then we need not recompute the answer to the query.
- Space savings are achieved from removing the irrelevant formulas.

2.3.4 Examples of Definitions

In this section we describe several instances of definitions in the space. We begin by showing how definitions in previous work can be couched in the space.

Other Definitions From the Literature

Subramanian investigates several definitions of irrelevance, which are all instances of weak irrelevance in our framework. The main definition investigated in [Subramanian, 1989] is the following:

Definition 2.11: Let ϕ be a formula, ψ be a query and Δ be a knowledge base. The formula ϕ is said to be irrelevant to ψ, denoted by $WI_1(\phi, \psi, \Delta)$ if there exists a subset Δ_1 of Δ, such that $\Delta_1 \not\models \phi$ and $\Delta_1 \models \psi$. ∎

This definition can be couched in our framework as follows:

Observation 2.12: For a complete set of inference rules S,

$$WI_1(\phi, \psi, \Delta) \equiv WI(\phi, \psi, \Delta, DI_3, \mathcal{D}_\psi).$$

Proof: Suppose $WI_1(\phi, \psi, \Delta)$ holds. Therefore, there is some subset Δ_1 of Δ such that $\Delta_1 \models \psi$ and $\Delta \not\models \phi$ and there is some derivation D of ψ from Δ_1. Clearly, $Base(D) \not\models \phi$, and consequently, $WI(\phi, \psi, \Delta, DI_3, \mathcal{D}_\psi)$ holds.

Conversely, assume $WI(\phi, \psi, \Delta, DI_3, \mathcal{D}_\psi)$ holds. Consequently, there is some derivation D of ψ from Δ such that $Base(D) \not\models \phi$. The KB consisting of $Base(D)$ is a subset of Δ and does not entail ϕ. Consequently, $WI_1(\phi, \psi, \Delta)$ holds. ∎

A variation of this definition that is described in [Subramanian and Genesereth, 1987] can be formulated as $WI(\phi, \psi, \Delta, DI_4, \mathcal{D}_\psi)$. Couching Subramanian's definitions in our framework highlights some of the properties of her definitions, mainly the fact that removing irrelevant formulas may not always lead to speeding up inference.

A definition of irrelevance is described in [Srivastava and Ramakrishnan, 1992]. Their definition is equivalent to strong irrelevance when DI_2 is quantified over the set of all derivations of the query, i.e., it is equivalent to $SI(\phi, \psi, \Delta, DI_2, \mathcal{D}_\psi)$.

Several resolution strategies are based on removing irrelevant clauses. For example, for refutation resolution, clauses containing pure literals[9] can be shown to be strongly irrelevant (with respect to DI_1 and $\mathcal{D}_\psi$), and can therefore be removed. Tautologies can be shown to be weakly irrelevant (with respect to DI_1 and $\mathcal{D}_\psi$) and therefore are removed by the *tautology elimination* strategy [Genesereth and Nilsson, 1987].

The question of detecting when a query is independent of an update is closely related to the notion of irrelevance. In Chapter 5, we show that definitions of independence investigated by Elkan [Elkan, 1990] and Blakeley et al [Blakeley *et al.*, 1989] are equivalent to weak irrelevance (specifically, $WI(\phi, \psi, \Delta, DI_1, \mathcal{D}_\psi)$). This observation enabled us to develop new algorithms for detecting independence.

Irrelevance with Minimal Derivations

Interesting definitions of irrelevance are obtained by considering cases in which DI is required to hold only for *minimal* derivations, i.e., where the choice $\mathcal{D}_0$ along the second axis is the set of minimal derivations. There are many ways of defining minimality of derivations. Here, we consider three possible definitions. Recall that

[9] A literal is pure if and only if it has no instance that is complementary to an instance of another literal in the knowledge base [Genesereth and Nilsson, 1987].

a derivation is a sequence $\alpha_1, \ldots, \alpha_n$, and it can be viewed as a tree formed by the subgoal relation. The following are three possible definitions of minimality.

> $M1$: A derivation D is minimal if does not have two identical formulas, α_i and α_j such that α_i is an ancestor of α_j.
>
> $M2$: A derivation D is minimal if whenever α_1 and α_2 are two identical nodes in the tree, their subtrees are identical. (essentially this means that if a formula is used in two places in the proof, then its derivation in both places is identical).
>
> $M3$: A derivation D is minimal if there is no other derivation of the query D' such that $Base(D') \subset Base(D)$ and $Base(D') \neq Bas(D)$.

To see the difference between the classes of derivations, consider Example 2.2 and assume we also had a rule
$r_5 : canTA(X, Y) \Rightarrow pass(X, Y)$.

Figure 2.2 shows three derivations. Derivation (a) is not a member of $M1$ because $pass(Fred, 202)$ is a subgoal of the query. Derivation (b) is a member of $M1$ but is not a member of $M2$ because $pass(Fred, 202)$ is derived in two different ways. Finally, derivation (c) is a member of $M1$ and $M2$, but not a member of $M3$ because the query can be derived using a subset of the base of the derivation (using only $passExam(Fred, 202)$ and the rules).

Note that $M1 \supsetneq M2$ but $M1 \not\supseteq M3$. Interestingly, the definitions of strong irrelevance for M1 and M2 turn out to be equivalent:

Lemma 2.13: *The definitions of strong irrelevance are equivalent for $M1$ and $M2$, i.e.,*

$$SI(\phi, \psi, \Sigma, DI_2, M1) \equiv SI(\phi, \psi, \Sigma, DI_2, M2).$$

Strong irrelevance for $M3$ is stronger than the other two, i.e.,

$$SI(\phi, \psi, \Sigma, DI_1, M3) \Rightarrow SI(\phi, \psi, \Sigma, DI_1, M2).$$

Proof: Since $M1 \supseteq M2$, it follows from Lemma 2.7 that

$$SI(\phi, \psi, \Sigma, DI_2, M1) \Rightarrow SI(\phi, \psi, \Sigma, DI_2, M2).$$

To show the converse, we show that if D is a derivation of ψ from a knowledge base $\Delta \in \Sigma$ such that $\phi \in D$ and $D \in M1$, then there is a derivation D' (of ψ from Δ), such that $\phi \in D'$ and $D' \in M2$. Consequently,

$$\neg SI(\phi, \psi, \Sigma, DI_2, M1) \Rightarrow \neg SI(\phi, \psi, \Sigma, DI_2, M2)$$

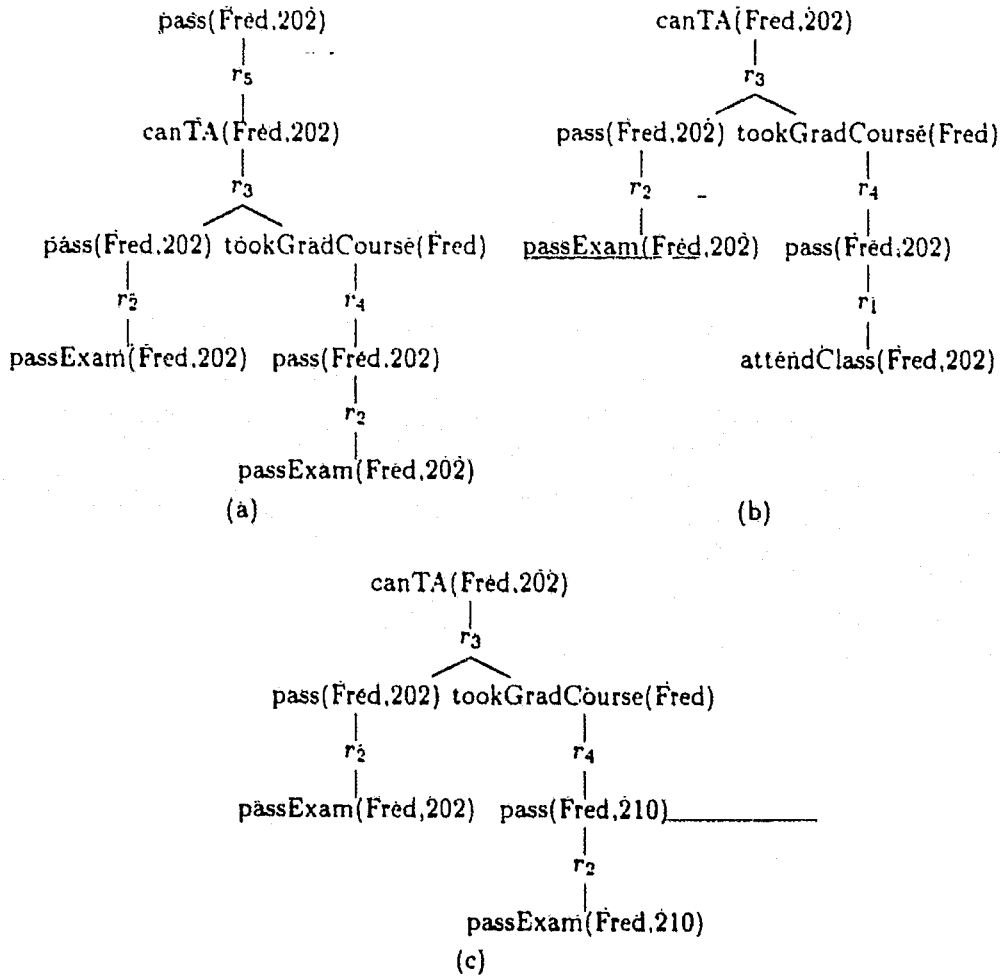

Figure 2.2: Minimal derivations

and therefore

$$SI(\phi, \psi, \Sigma, DI_2, M2) \Rightarrow SI(\phi, \psi, \Sigma, DI_2, M1).$$

Let D a derivation such that $\phi \in D$ and $D \in M1$. Suppose τ is a formula that appears twice (or more) in D with non identical subtrees, T_1 and T_2. Let T be one of these subtrees in which ϕ appears (if ϕ does not appear in either T_1 or T_2 then choose one arbitrarily). Replace all the subtrees of τ in D by T. Note, that since $\phi \in M1$, this transformation is well defined. Denote the resulting derivation by D'. The derivation D' is a valid derivation of the query, it includes ϕ and furthermore, τ has a unique subtree in every appearance. We repeat this transformation until we cannot find a formula τ which appears with two (or more) non identical derivation subtrees. The resulting derivation will be a member of $M2$ and will include ϕ. Note that the number

of transformations must be finite, because the number of transformations we perform is at most the number of distinct formulas in D.

To show that strong irrelevance for $M3$ entails the other two, we show the following. Let D be a derivation such that $\phi \in Base(D)$ and $D \in M3$. We construct a derivation D' such that $\phi \notin Base(D')$ and $D' \in M1$ as follows. For every pair of identical nodes $\tau_1, \tau_2 \in D$ such that τ_1 is an ancestor of τ_2, we replace the subtree of τ_1 by the subtree of τ_2.[10] The resulting derivation D' is a member of $M1$. Moreover, if $\phi \in Base(D)$ then $\phi \in Base(D')$, otherwise, D would not be a member of $M3$. As before, this construction shows that

$$\neg SI(\phi, \psi, \Sigma, DI_1, M3) \Rightarrow \neg SI(\phi, \psi, \Sigma, DI_1, M1)$$

and therefore

$$SI(\phi, \psi, \Sigma, DI_1, M1) \Rightarrow SI(\phi, \psi, \Sigma, DI_1, M3).$$

∎

It should be noted that removing formulas that do not appear in minimal derivations (of the type $M1$) will speed up inference for many search strategies employed by inference engines. For example, an inference mechanism performing depth-first search or breadth first search will always find a derivation of the query that belongs to $M1$ before it finds one that does not.

In Chapter 3, we describe an algorithm for automatically deciding which formulas are strongly irrelevant to a query when considering $M1$ (and therefore also $M2$) for Horn rule theories. We also show that deciding which formulas are strongly irrelevant for $M3$ is undecidable in general. However, Lemma 2.13 implies that the algorithms of Chapter 3 provide a sufficient condition for strong irrelevance for $M3$.

Relationship to Truth Maintenance Systems

Strong irrelevance for $M3$ can be characterized in terms of labels in an assumption based truth maintenance system (ATMS) [de Kleer, 1986]:

Observation 2.14: Assume a complete set of inference rules and let ϕ be a formula and ψ be a query. $SI(\phi, \psi, \Delta, DI_1, M3)$ holds if and only if ϕ does not appear in any ATMS label of ψ.

Proof: An ATMS label of ψ is a set of support S such that $S \models \psi$ and such that there does not exist a subset $S' \subset S$ such that $S' \models \psi$. Clearly, $SI(\phi, \psi, \Delta, DI_1, M3)$ does not hold if and only if there is some derivation D such that $Base(D)$ is a minimal support set for ψ. The set $Base(D)$ will be the ATMS label for ψ. ∎

[10] If there are several such pairs, we do so in an arbitrary order.

This observation shows that even though finding all formulas that cannot appear in the ATMS labels of a query is undecidable, the algorithms that we present for deciding strong irrelevance can be used to prune formulas from consideration when computing ATMS labels.

Evidence Based Definitions Revisited

As described in the beginning of this chapter, much of the previous analysis of irrelevance was done in a slightly different context. Specifically, it has addressed the following question:

- Given a set of *evidence* $\mathcal{E}$ and query q, which formulas are irrelevant to q with respect to the evidence?

As stated, our analysis differs in that we want to know which part of our knowledge base (that can be viewed as evidence) is irrelevant to the query. To reconcile the two, we can ask the following question:

- Given a knowledge base Δ and a subset of it $\mathcal{E}$ called the evidence, which parts of Δ are irrelevant to q, given the evidence?

Intuitively, the formulas in $\mathcal{E}$ are basic assumptions about the domain that we want to use if possible. We can define such a notion in our framework in several ways. One way is to limit the set of derivations considered to those in which formulas in $\mathcal{E}$, if they appear in a derivation D, must be in $Base(D)$. This means that we do not allow evidence to be derived from other formulas. A slightly stronger way of formalizing this notion is by considering derivations of the query that have minimal support with respect to the evidence $\mathcal{E}$, denoted by $\mathcal{D}_{\mathcal{E}}$, as follows:

Definition 2.15: A derivation D is said to have minimal support w.r.t. the evidence $\mathcal{E}$ if there does not exist a strict subset $S \subseteq Base(D)$ such that $S \cup \mathcal{E} \vdash \psi$. ∎

Intuitively, derivations in $\mathcal{D}_{\mathcal{E}}$ use the formulas in the evidence as much as possible.

Returning to our example, if the query is $q \doteq canTA(Fred, 101)$ and the set of evidence is empty, then the atom $passExam(Fred, 101)$ is not strongly irrelevant to the query. However, if our evidence includes the atom $\{pass(Fred, 101)\}$, then $passExam(Fred, 101)$ becomes strongly irrelevant to q.

2.4 Automatically Deriving Irrelevance Claims

A key question that we address in this dissertation is how (and whether) irrelevance claims can be derived automatically. Specifically, we are interested in two problems.

First, given a knowledge base, a query and a specific definition of irrelevance, we want to find automatically which formulas in the knowledge base are irrelevant to the query. Second, given an irrelevance claim, we want to derive other irrelevance claims that logically follow. We focus on solving the first problem. In Chapter 4 we show how results pertaining to the first problem can be used to solve the second.

In general, deciding which formulas are irrelevant to a given query will be more expensive than answering the query itself, especially in large knowledge bases. Furthermore, if the knowledge base changes, the relevance reasoning needs to be repeated. In order for our algorithms to be of practical interest, we will derive irrelevance claims by examining only a small and stable part of the KB, and will derive irrelevance claims that hold independent of any changes that are made to other unexamined parts. Furthermore, our irrelevance claims will hold for a family of queries (given by a query schema).

We examine the question of automatically deriving irrelevance claims for Horn knowledge bases that consist of a set of Horn rules $\mathcal{P}$ and a set of ground atomic facts G. We distinguish two sets of predicates in the KB: *base predicates* (often called EDB predicates) and *derived predicates* (IDB predicates). The base predicates are those that appear in the ground facts of G. The derived predicates are those that appear in the consequents of the rules. For syntactic convenience, we assume that base predicates do not appear in the consequents of rules. The KB consisting of Given a set of rules $\mathcal{P}$ and ground facts G, can also be viewed as defining relations for the derived predicates in terms of the base predicates.

Many of the interactions between rules in a knowledge base can be deduced by considering the semantics of some of the predicates that appear in them, such as order predicates ($=$, $\neq$, $\leq$, $<$, $\geq$, $>$) or sort predicates. For instance, in Example 2.2, g_3 and g_4 were deemed strongly irrelevant by considering the semantics of the predicate $\leq$. We therefore distinguish a subset of the predicates which we name *constraint predicates* (or *interpreted predicates*). These predicates will be treated much like the EDB predicates, with the difference being that their semantics will be enforced in our relevance reasoning. A *constraint formula* is a formula in some language $\mathcal{L}$ for expressing constraints that involves only literals of constraint predicates and logical connectives (e.g., disjunction, conjunction, negation). For example the formula $Even(x) \wedge (x > 100)$ is a constraint if the predicate $Even$ is a sort predicate. We place few restrictions on the properties that the constraint predicates need to satisfy. Formally, a formula f (in the language $\mathcal{L}$), with free variables $X_1, \ldots, X_n$, describes a (possibly infinite) relation $R_f(X_1, \ldots, X_n)$, which is the set of all tuples satisfying the constraints expressed by f. We assume the following properties of constraint formulas:

Closure: Given formulas f_1 and f_2, it is possible to effectively construct formulas that express:

- The join of R_{f_1} and R_{f_2}.
- A projection of R_{f_1} (i.e., a relation consisting of only a subset of the arguments of R_{f_1}).
- A selection $\sigma_{i=j}R_{f_1}$, where i and j are some columns of R_{f_1} (i.e., a relation consisting of only tuples in which columns i and j are equal).
- A selection $\sigma_{i=c}R_{f_1}$, where i is some column of R_{f_1} and c is a constant in the language $\mathcal{L}$.

Equivalence: Given formulas f_1 and f_2, it is decidable whether $R_{f_1} = R_{f_2}$.

Satisfiability: Given a formula f, it is decidable whether R_f is nonempty.[11]

Finiteness: Let C be a finite set of constants in the language $\mathcal{L}$, and let $\mathcal{F}$ be a finite set of formulas in the language $\mathcal{L}$ that have at most n free variables (for some fixed n) and only constants from C. Then applications of the operators (discussed in the Closure Property) to formulas in $\mathcal{F}$ may create only a finite number of nonequivalent formulas over n (or fewer) free variables.

Moreover, if f is a formula with a free variable X, then f can imply $X = c$, where c is a constant of the language $\mathcal{L}$, only if c appears in f.

The Closure condition guarantees that we can perform the basic manipulations of the constraints within our constraint language. The second an third conditions guarantee that we can identify two equivalent constraints. The Finiteness constraint guarantees that we only have a finite number of non isomorphic constraints. In Chapter 3, we discuss the case in which the Finiteness condition does not hold. The procedures needed to compute the closure operations, equivalence and satisfiability are assumed to be given.[12]

These conditions cover a wide class of interpreted constraints. The following are a few examples:

- **Order constraints:** The language consisting of the predicates $=$, $\neq$, $\leq$, $<$, $\geq$, $>$ and the connectives $\wedge$ and $\vee$. If we allow only conjunctions, the Closure condition will not be satisfied. This special case is treated in Section 3.2.1.
- **Sort constraints:** A constraint language based on a finite sort hierarchy, and the connectives $\wedge, \vee$ and $\neg$.

[11] Note that if we have a formula FALSE in our language, denoting the empty relation, then the Satisfiability Property will follow from the Equivalence Property.

[12] Typically, these procedures are efficient. For example, for order constraints, testing equivalence is cubic in the number of variables.

- **Finite, given relation:** Often, a given relation that is relatively small and stable can be best viewed as a constraint. Any given finite relation satisfies the properties that we require.

Hereafter, a *constraint* will refer to a constraint formula in some constraint language $\mathcal{L}$.

Finally, we also consider cases in which the rules contain negated literals in their antecedents (and are therefore no longer Horn). In such cases, we assume:

- The negation is stratified [Ullman, 1989].[13]
- The negation is *safe*, i.e., if a variable appears in a negative literal in the antecedent then it also appears in a positive literal in the antecedent.

We consider queries that are atoms which are either ground (i.e., is $p(a)$ entailed by $\mathcal{P} \cup G$?), or contain free variables (i.e., find some, or all, x such that $p(X)$ is entailed by $\mathcal{P} \cup G$).

In many applications using Horn rule knowledge bases, it is the case that the bulk of the KB is ground facts, and the ground facts are much more prone to frequent changes than the remainder of the KB. Therefore, we address the irrelevance problem for the set of knowledge bases that differ only on ground facts. Specifically, we address the following question. Let $\mathcal{P}$ be a set of rules, and let $\Sigma_{\mathcal{P}}$ be the set of knowledge bases of the form $\mathcal{P} \cup G$, where G is a set of ground atomic facts for the EDB predicates. The question then is whether we can decide whether a given fact ϕ is irrelevant to the query ψ, i.e., does $SI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D})$ or $WI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D})$ hold. Note that in Horn rule KBs, DI_1 and DI_2 are equivalent for the rules and the EDB formulas. For IDB formulas, DI_1 is trivially true. Therefore, we consider the definition DI_2 in our investigations.

A summary of the decidability results pertaining to this question is shown in Table 2.1. As we prove below, weak irrelevance is undecidable whenever the rules contain recursion. In contrast, strong irrelevance is efficiently decidable for a larger class of languages. In Chapter 3 we present algorithms for deriving irrelevance for several cases of strong-irrelevance, including irrelevance under minimal derivations. Chapter 5 describes an algorithm for detecting weak irrelevance in the presence of constraints. In the next section we prove a few undecidable cases of irrelevance.

The complexity of the algorithms we describe are all linear in the number of rules in the KB and do not depend on the number of ground facts. The complexity is exponential in the arity of predicates. When we consider irrelevance under minimal derivations, the algorithms are doubly exponential in the arity of the predicates.

[13]The rules are stratified if there are no dependency cycles that involve negations between the predicates in the KB. The dependency graph of the KB has one node for every predicate and there is an arc from p to q if p appears in the antecedent of a rule whose consequent is q.

However, arities of predicates tend to be very small (e.g., frame systems usually employ mostly binary predicates). Furthermore, we believe that exponential running time is not likely to occur in practice (since finding examples with exponential running time requires careful crafting of the rules),[14] and so, the algorithms we present will be efficient in practice.

Language	Strong Irrelevance			Weak Irrelevance
	All Derivations	Minimal Derivations	Minimal Support Derivations	All Derivations
Horn rules with no recursion	**Decidable** Follows from [Kifer, 1988]		**Decidable** Follows from [Sagiv, 1988]	
No recursion + constraints	**Decidable** Follows from Chapter 3		**Decidable** Chapter 5	
Datalog	**Decidable** Chapter 3		**Undecidable** Lemma 2.17	Lemma 2.16
Datalog with constraints	**Decidable** Chapter 3		**Undecidable** Lemma 2.17	Lemma 2.16
General Horn rules	**Undecidable** Follows from [Abiteboul and Hull, 1988].			
Datalog with Stratified Negation	**Undecidable** Lemma 2.18		Lemma 2.17	Lemma 2.16
Negated base predicates	**Decidable** Section 3.4		**Undecidable** Lemma 2.17	Lemma 2.16
Unary base predicates	**Decidable** [Levy *et al.*, 1993]			

Table 2.1: Decidability of deriving irrelevance claims

2.4.1 A Few Undecidable Cases

The following shows that weak irrelevance is undecidable even for function-free Horn rules (i.e., datalog):

Lemma 2.16: *Let $\mathcal{P}$ be a set of datalog rules and ψ be a query. Determining whether $WI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_{\psi})$ holds is undecidable even if the rules have no interpreted predicates.*

Proof: Let $r \in \mathcal{P}$ and ψ be a query. We prove the lemma by showing that the claim $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_{\psi})$ holds if and only if r is redundant, i.e., the set of rules $\mathcal{P} - r$ is equivalent to the set $\mathcal{P}$. In proof, suppose that $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_{\psi})$ holds, then for any knowledge base $\Delta \in \Sigma_{\mathcal{P}}$, $WI(r, \psi, \Delta, DI_2, \mathcal{D}_{\psi})$ holds. Therefore, if there is a derivation of ψ, then there is one that does not use r. Consequently, r can be

[14] Specifically, it requires rules that create in their consequents permutations of the variables from their antecedents

removed from $\mathcal{P}$ without changing the answer to ψ, regardless of the ground facts in the KB, and therefore, r is redundant. Conversely, if r is redundant, that means that for every $\Delta \in \Sigma_{\mathcal{P}}$, if ψ is provable, there is a derivation that doesn't contain r. Therefore, $WI(r, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_\psi)$ holds.

However, it follows from [Shmueli, 1987] that redundancy is undecidable for datalog theories. Therefore, weak irrelevance is undecidable. ∎

It should be noted that Subramanian [Subramanian, 1989] states a similar result for DI_3, but does not give a proof. The following lemma shows that strong irrelevance under minimal-support derivations is also undecidable:

Lemma 2.17: *Determining whether $SI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, M3)$ holds is undecidable for datalog knowledge bases without interpreted predicates.*

Proof: We prove the lemma by reducing the containment problem of datalog programs to the strong irrelevance problem for $M3$. Since it follows from [Shmueli, 1987] that containment is undecidable, strong irrelevance for $M3$ is also undecidable.

Let P_1 and P_2 be two datalog programs. Let e be a new EDB predicate appearing nowhere in P_1 or P_2. We construct a program P_3 as follows:

$p_1(X) \wedge e(X) \Rightarrow p_3(X)$
$p_2(X) \Rightarrow p_3(X)$

We show that $SI(e(X), p_3(X), \Sigma_{P_3}, DI_2, M3)$ holds if and only if $P_1 \subseteq P_2$. Suppose $P_1 \subseteq P_2$ holds. We show that for any given constant a, $e(a)$ cannot be part of a minimal-support derivation of $p_3(a)$. Suppose G is a database from which $e(a)$ is part of a minimal support derivation D of $p_3(a)$. We can assume that G contains only the ground atoms in $Base(D)$. The database $G - e(a)$ is therefore enough to derive $p_1(a)$. However, since $P_1 \subseteq P_2$, the database $G - e(a)$ is also enough for deriving $p_2(a)$, and therefore, $p_3(a)$. However, this would mean that D is not a minimal support derivation because the derivation of $p_3(a)$ through $p_2(a)$ uses a strict subset of $Base(D)$.

Conversely, suppose $P_1 \not\subseteq P_2$. Therefore, there is a database G and a constant a such that $p_1(a) \in P_1(G)$ and $p_2(a) \notin P_2(G)$. Consider the database $G \cup e(a)$, and let D be a minimal-support derivation of $p_1(a)$. The formula $e(a)$ will now be part of a minimal support derivation of $p_3(a)$, constructed from D and $e_3(a)$, using the first rule. Consequently, $SI(e(X), p_3(X), \Sigma_{P_3}, DI_2, M3)$ does not hold. ∎

Finally, we show that strong irrelevance is undecidable when we allow the rules to have stratified negation. In our discussion, we assume perfect model semantics of the rules (cf. [Ullman, 1989]).[15]

[15] The perfect model of a set of rules is the one computed in a bottom-up fashion, stratum by stratum.

Lemma 2.18: *Let $\mathcal{P}$ be a set of datalog rules with stratified negation and $r \in \mathcal{P}$. Determining whether $SI(r, \mathcal{V}, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_{\mathcal{V}})$ is undecidable, even if $\mathcal{P}$ has no interpreted predicates.*

Proof: Testing equivalence of two datalog programs is undecidable [Shmueli, 1987]. We will reduce the equivalence problem to the irrelevance problem of a rule in a stratified program, i.e., we show that if there is an algorithm for deciding whether a rule r in a datalog knowledge base $\mathcal{P}$ is strongly irrelevant, then we can design an algorithm for testing equivalence of two programs.

Let $\mathcal{P}_1$ and $\mathcal{P}_2$ be two programs with query-predicates p_1 and p_2.[16] Without loss of generality we can assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ have distinct sets of IDB predicates. To test equivalence, it is enough to test whether $\mathcal{P}_1 \supseteq \mathcal{P}_2$ and $\mathcal{P}_2 \supseteq \mathcal{P}_1$. Let Q be the program containing the rules of $\mathcal{P}_1$ and $\mathcal{P}_2$ and the rule

$$r : p_1(\bar{X}) \wedge \neg p_2(\bar{X}) \Rightarrow q(\bar{X}).$$

where q is the query predicate of Q and it appears nowhere in P_1 or P_2. Note that Q is a stratified program, since r is the only rule containing negation. Clearly, r is strongly irrelevant to q if and only if $\mathcal{P}_2 \supseteq \mathcal{P}_1$, since r will be used in a derivation if and only if there is some database in which some ground tuple is a member of p_1 and not of p_2. In a similar fashion, we can create a program with a rule r' which will be strongly irrelevant if and only if $\mathcal{P}_1 \supseteq \mathcal{P}_2$. Consequently, if rule irrelevance is decidable for programs with stratified negation, then program equivalence will be decidable. ∎

Chapter 3 and [Levy *et al.*, 1993] describe restrictions on stratified negation in which strong irrelevance is still decidable.

2.5 Summary and Related Work

We have presented a general framework for analyzing and comparing definitions of irrelevance. The framework is based on a proof-theoretic analysis of the notion of irrelevance, and therefore enables us to address the two key issues in relevance reasoning: automatically deriving irrelevance claims and the utility of removing irrelevant formulas. Aside from suggesting new definitions of irrelevance, the framework encompasses previous definitions that were discussed in the literature. For example, as will be discussed in Chapter 5, the framework sheds new light on the problem of detecting independence of queries from updates. Within the framework, we have identified a class of irrelevance claims, namely strong irrelevance, which have several

[16] Recall that the programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are equivalent if for any database, the relation computed for p_1 is the same as that computed for p_2.

desirable properties. First, removing strongly irrelevant formulas is guaranteed never to slow inference (and usually speed it up significantly). In Chapter 4 we will present experimental results to validate the impact of these speedups. Second, we demonstrate in Chapter 3 that for some languages, it is possible to efficiently decide which formulas are strongly irrelevant to a given query. Finally, strong irrelevance satisfies several properties that have been argued to be natural for the common sense notion of irrelevance (such as closure under union and some forms of monotonicity).

The notion of irrelevance has been formally investigated in the philosophy literature [Keynes, 1921; Carnap, 1950; Gärdenfors, 1978]. As stated earlier, the focus of the discussion there was on formalizing a notion of irrelevance that would fit common sense notions of the word. The discussion did not concern itself with the computational aspects of reasoning about irrelevance, as we focus on here. Moreover, the focus of the discussion in that literature is on analyzing irrelevance w.r.t. a set of evidence, which is usually treated as a closed theory (i.e., independent of changes in the form of the formulas or the inference mechanism). In our analysis, we are concerned with finding irrelevant formulas in a large KB, where the form of the KB and the inference mechanism play key roles.

A related concept discussed in the formal logic community is of *relevance logics* (e.g., [Anderson and Belnap, 1975; Dunn, 1986; Avron, 1992]). The key idea in relevance logics is to modify the logic and the inference rules such that only *relevant implications* can be made. However, two issues are still largely open in this field. The first is devising clean and intuitive semantics for these logics, and the second is providing tractable inference for them. In contrast, our analysis of irrelevance assumes that the underlying logic remains unchanged.

Within AI, the notion of irrelevance was used rather informally in various works, such as RLL [Greiner, 1980] and compositional modeling [Falkenhainer and Forbus, 1991]. Irrelevance was investigated extensively in the context of probabilistic reasoning [Pearl, 1988]. However, in that context, irrelevance has a natural definition based on the notion of conditional independence. This notion does not carry over to the context of logical knowledge bases.

The work most related to ours is the analysis of irrelevance given by Subramanian [Subramanian and Genesereth, 1987; Subramanian, 1989]. Subramanian's motivations for analyzing irrelevance are similar to ours, namely, reformulating the knowledge base to create one that is simpler and will therefore lead to more efficient inference. However, her framework does not provide sufficient distinctions to enable one to analyze the issues of deriving irrelevance claims and the utility of doing so. Our framework can be viewed as a refinement of hers, where in addition to considering the form of the KB, we also consider the possible derivations that an inference mechanism can pursue. The specific definitions that she considers are formulated in our framework as variations of weak irrelevance. Subramanian also defined a class of

computational-irrelevance claims whose exploitation leads to computational savings, but only gave some straightforward examples of such claims. Our class of strong irrelevance claims is a prime example of computational-irrelevance claims.

It should be noted that in [Subramanian and Genesereth, 1987], a definition of strong-irrelevance is given. However, instances satisfying this definition are not necessarily instances of computational irrelevance. For instance, under her definition, in Example 2.2, the atoms g_1-g_4 are also strongly irrelevant to the query $canTA(Fred, 101)$. Finally, Subramanian discusses several algorithms for detecting irrelevance. However, they focus on the case of propositional logic KBs and require solving the query as part of the algorithm. Consequently, their utility is questionable. She considers an extension of the algorithm to the first order case, using the concept of a *definability graph*. This graph denotes only the dependencies between predicates in the KB, and therefore does not enable relevance reasoning beyond simple reachability tests.

Chapter 3

The Query-tree

In the previous chapter we posed the problem of automatically deriving irrelevance claims. This chapter describes algorithms for automatically deriving strong irrelevance claims. Recall that a formula is strongly irrelevant to a query if some condition (DI) holds for *all* the derivations in some set $\mathcal{D}_0$ of derivations of the query. Therefore, in order to deem a formula strongly irrelevant, we need to meet two challenges. The first is to establish properties of a possibly infinite set of derivations by a finite procedure. The second is that even if there is only a finite number of derivations, an algorithm that actually enumerates all of them will be of little interest, both theoretically and in practice. Therefore, we would like an efficient method of establishing properties of a set of derivations without actually enumerating them.

This chapter describes a novel tool, the *query-tree*, (see example in Figure 3.2) that is used to establish efficiently the properties of a set of derivations. The query-tree is a data structure that encodes a (possibly infinite) set of derivations so that properties of that set can be established by inspecting the tree. For example, by inspecting the query-tree we can check whether a certain formula can be part of some derivation of the query, and therefore decide whether it is strongly irrelevant to the query. The query-tree is a general method for encoding a given set of derivations. Query-trees differ depending on which set of derivations we want the tree to encode. The challenge in building a query-tree is to ensure that it encodes *all* and *only* the derivations in which we are interested. When it does, inspecting the query-tree is akin to inspecting the entire set of derivations.

We begin in Section 3.1 by describing the principles underlying the query-tree method. We present a general method for building a query-tree that encodes a desired set of derivations. In the subsequent sections describe several instances of the query-tree, obtained by following the general method. Section 3.2 considers the problem of building a query-tree for function-free Horn-rule knowledge bases with interpreted predicates (in this chapter we assume that the interpreted predicates satisfy

the conditions given in Section 2.4). We show how to build a query-tree that encodes precisely the set of possible derivations of the query. It also discusses extending the algorithm to the case where rules may have function symbols. In Section 3.3 we describe how to build a query-tree that encodes only the set of *minimal* derivations of the query. Section 3.4 considers an extension beyond Horn rule knowledge bases. We show how to build a query-tree that encodes precisely the set of derivations of the query when the rules have negated EDB literals in the antecedent.

3.1 The Query-tree Method

3.1.1 Symbolic Derivations

In the context of Horn rule knowledge bases, we view a derivation as a tree consisting of goal-nodes and rule-nodes (see Figure 3.1(a)). The root of the tree is a goal-node containing the query atom. If a goal-node g was derived using a rule r and the antecedents $g_1, \ldots, g_m$, then r is the child of g and its children are $g_1, \ldots, g_m$. The leaves of a derivation are ground atomic facts from the database.

Since the query-tree will be built based only on the rules in the knowledge base (without looking at the ground atomic formulas), it will encode a set of derivations by encoding a set of *symbolic derivations* (see Figure 3.1(b)). Like a derivation, the root of a symbolic derivation tree is a goal-node of the query atom (which does not have to be ground). The child of a goal-node is a rule-node containing a rule whose consequent unifies with the goal-node. The rule-node has a goal-node child for every conjunct in its antecedent, and the contents of each such goal-node is the corresponding conjunct in the unification of the rule with g. The leaves of a symbolic derivation tree contain atoms of EDB predicates or atoms of interpreted predicates. A symbolic derivation tree contains only variables and constants that appear in the rules. If a rule-node r' in a symbolic derivation tree contains the rule r from the knowledge base, we say that r' is a *rule of* r. Similarly, if g is a goal-node containing an atom of the predicate p, we say that g is a node of p. We assume that the variable patterns in a symbolic derivation tree implicitly represent all the equalities implied by the interpreted constraints, i.e., if the conjunction of the interpreted constraints in the rules imply that two variables X and Y must be equal,[1] then the same variable appears in all the positions of X and Y.

A symbolic derivation represents the set of derivations that can be obtained by assigning constants to the variables in the derivation. Therefore, a set of symbolic derivations represents the union of derivations represented by each element of the set.

In order to build a query-tree that enables us to establish some property of a set

[1] For example, one rule contains the literal $X \leq Y$ and the other contains $X \geq Y$.

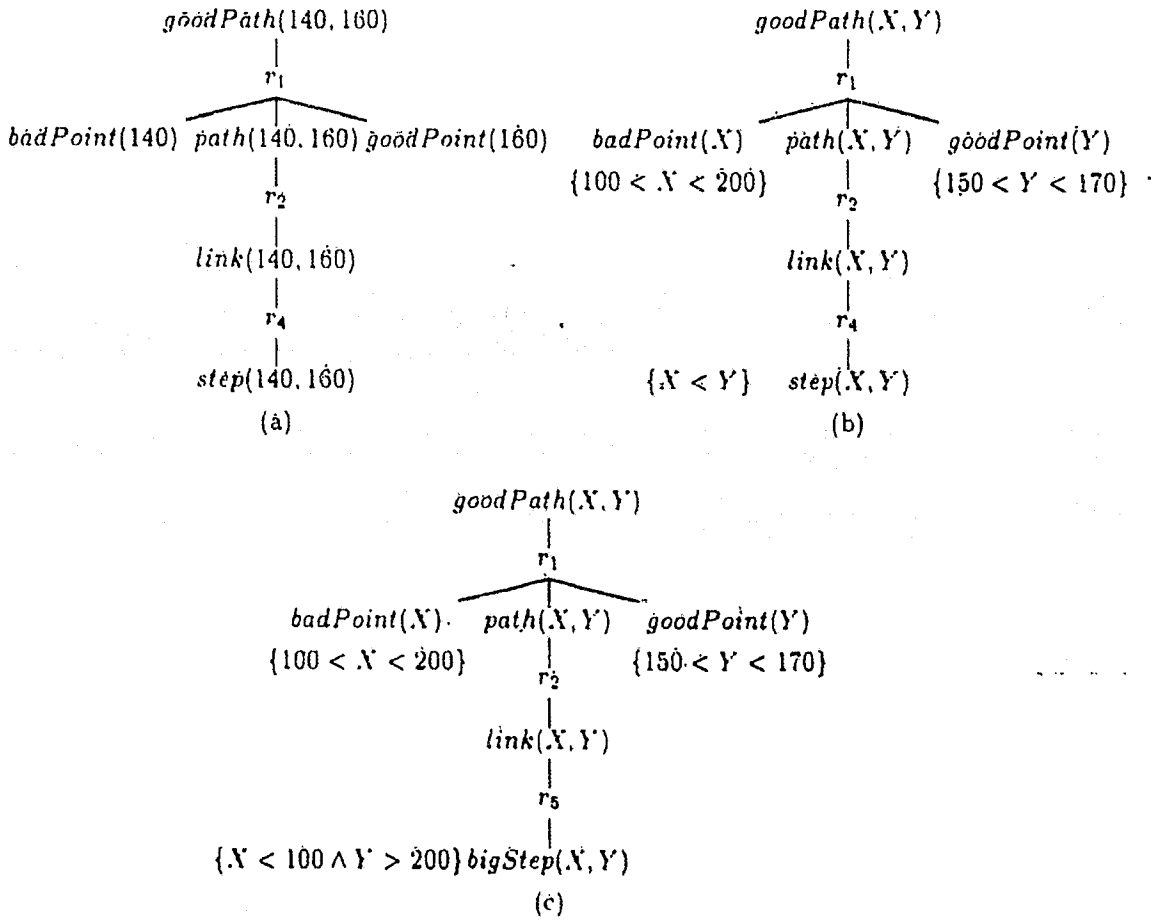

Figure 3.1: (a) is a ground derivation. (b) is a satisfiable symbolic derivation and (c) is an unsatisfiable symbolic derivation.

of derivations $\mathcal{D}$, we first identify a set of symbolic derivations Π, such that encoding the set Π will enable us to deduce the properties we need about $\mathcal{D}$. For example, if we are building a query-tree to encode all derivations of the query when interpreted predicates may exist in the rules, the set Π will be the symbolic derivations with the property that the interpreted constraints on the variables in the derivation are satisfiable. We denote this set by Π_{sat}. For example, in Figure 3.1, considering the semantics of the order predicates implies that derivation (b) is satisfiable, while derivation (c) is not satisfiable. Given such a set Π, our goal will be to build a query-tree that encodes precisely the symbolic derivations in Π. In our discussion, we use Π both to denote a set of symbolic derivations or to denote the property that distinguishes symbolic derivations in the set.

In this chapter, we consider properties Π on symbolic derivations that can be recognized by *finite labeling*. Formally, this means the following:

- There is a finite number of labels (of finite size) that can be attached to nodes of symbolic derivation trees. The number of such labels depends only on the size of the knowledge base (and not on the size of the symbolic derivation).
- The label of a node is computable from the labels of its children (or vice versa).
- Whether a symbolic derivation tree d satisfies property Π can be computed from the labels of d. Specifically, we distinguish one label called the *inconsistent* label. It should be the case that a symbolic derivation tree d has the property Π if none of its nodes has the inconsistent label.

Essentially, the finite labeling condition means that the set of symbolic derivation trees in Π can be recognized by a finite tree automaton.[2] The query-tree can be viewed as a recognizer for these symbolic derivations. The first condition assures that the number of states in the automaton is finite and therefore that we will be able to identify Π using a finite structure. The second condition guarantees that we can specify the transitions of the automaton. Specifically, this means that given an input symbol and the current state, the next state can be determined by inspecting the current state *alone* and not by inspecting the entire path that led to the current state. Finally, the third condition guarantees that examining the labels is indeed sufficient to recognize symbolic derivations that satisfy Π. We assume that the labels completely specify the equality relations on the variables in the node, that are entailed by the interpreted constraints in the rules.

Returning to our example, to encode the set of symbolic derivations Π_{sat}, the label of a node will be a *constraint-label* describing the constraints that the instances of that node must satisfy. Note that because we require the constraint language to satisfy the Finiteness Property (Section 2.4), the number of non-equivalent labels will be finite. A symbolic derivation will be satisfiable if and only if the constraint labels of the nodes are satisfiable.

3.1.2 Building A Query-tree

The query-tree is a symbolic AND-OR tree (a.k.a. rule-goal tree). It consists of goal-nodes and rule-nodes (see Figure 3.2). The root of the tree is a goal-node containing the atom of the query. Each child of a goal-node containing g is a rule-node, containing a rule from the knowledge base, whose consequent unifies with g. The rule-node has a goal-node child for every conjunct in its antecedent, and the contents of each such

[2] See [Slutzki, 1985] for an exposition of tree automata.

goal-node is the corresponding conjunct in the unification of the rule with g. Unlike symbolic derivation trees, a rule-node in the query-tree will *not* have a child for an antecedent of an interpreted predicate.[3]

The knowledge base Δ consists of the following rules:

$r_1 : badPoint(X) \wedge path(X,Y) \wedge goodPoint(Y) \Rightarrow goodPath(X,Y).$
$r_2 : link(X,Y) \Rightarrow path(X,Y).$
$r_3 : link(X,Z) \wedge path(Z,Y) \Rightarrow path(X,Y).$
$r_4 : step(X,Y) \Rightarrow link(X,Y).$
$r_5 : bigStep(X,Y) \Rightarrow link(X,Y).$

The following constraints are given on the ground facts:

$badPoint(X) \Rightarrow 100 < X < 200.$
$step(X,Y) \Rightarrow X < Y.$
$goodPoint(X) \Rightarrow 150 < X < 170.$
$bigStep(X,Y) \Rightarrow X < 100 \wedge Y > 200.$

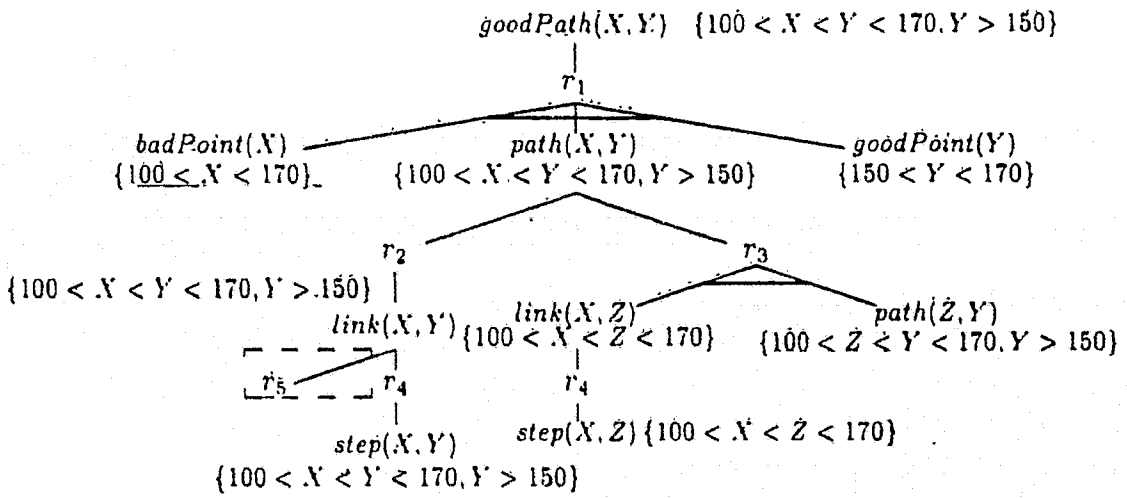

Figure 3.2: An example query-tree. Note that the rule r_5 is not expanded because it would result in an inconsistent constraint label. The expanded equivalent of the node $path(Z,Y)$ is $path(X,Y)$.

There are two key issues in building a query-tree. First, if a knowledge base has recursive rules, a simple minded top-down construction of the query-tree will not terminate. Second, we want to guarantee that the query-tree encodes precisely the set of symbolic derivations that satisfy the property Π. Therefore, we need some principled method for terminating the construction of the tree by not expanding some of the nodes. We do this by attaching *labels* to nodes in the tree (ultimately, these will be

[3]The constraints implied by the interpreted predicates will be reflected in the labels of the nodes, described shortly.

the same labels we use in showing that Π can be recognized by a finite-labeling). The labels partition the possible goal-nodes that can appear in the tree into equivalence classes. Two nodes are considered equivalent if there is an isomorphism between them and between their labels (where the isomorphism is defined by a mapping on the variables of the nodes). Based on the labels, we use the following termination condition. A goal-node g will not be further expanded if:

- g is a node of an EDB predicate, or
- There are no rules that can be unified with g, or
- Expanding the node g with a rule r will result in a child node with the *inconsistent* label, or
- There is some other goal-node in the tree g_1 such that g and g_1 are equivalent and such that g_1 has already been expanded.[4] We refer to g_1 as the *expanded equivalent* of g, denoted by $Eq(g)$.

Intuitively, there is no need to expand both g and g_1 because the labeling scheme guarantees that the subtrees that would appear under the g are precisely the ones that would appear under g_1. If a node g has an inconsistent label, it means that this node cannot appear in derivations that satisfy Π. For example, in Figure 3.2, expanding the rule r_5 will result in an inconsistent (i.e., unsatisfiable) constraint label.

In order to complete the specification of an algorithm for creating a query-tree, we need some method for assigning labels to nodes in the tree. Of course, the method must guarantee that the resulting query-tree encodes the desired set of derivations. The specific methods are described in the subsequent sections. Each method specifies three components:

1. An initial label c_0 for the root of the query-tree.
2. A function $TDLabel(r, c, g)$ that accepts a goal-node g with label c and a rule r that unifies with g and returns a label for the resulting rule-node child of g.
3. A function $TDproj(r, \theta, c, g)$ that accepts a label c for a rule-node containing a rule r that was unified with its father using a unifier θ, and a literal g in the antecedent of r, and returns a label for the goal-node child corresponding to g.

Given these functions, a query-tree can be built in two steps. In the first, the tree is expanded in a top-down fashion, using the above termination condition and the labeling procedure. In the second step, we *shake* the tree by removing all the nodes that are not reachable from the base predicates and from the root.[5] The details of the two steps are shown in Figures 3.3 and 3.4.

[4] Note that g_1 can be any node in the tree, not necessarily an ancestor of g.

[5] This step is needed because if a node in the query-tree is not reachable from the EDB leaves, it

procedure build-tree($\mathcal{P}, q, c_0$)
begin
/* Creating a query-tree T for the rules $\mathcal{P}$ and query q. */
/* The label of a node n in the query-tree is $c_f(n)$. */
The root of T is q with the label c_0.
repeat
  Let g be a node of an IDB predicate in T with label $c_f(g)$.
  if there is a node g_1 in T such that $g \equiv g_1$ and $c_f(g_1) \equiv c_f(g)$ **then**
    Set $Eq(g) = g_1$.
  else
    for each rule $r \in \mathcal{P}$ **do**
      if rule r unifies with g **then**
        θ = the most general unifier of r and g.
        $c = TDLabel(r, c_f(g), g)$
        if c is not *inconsistent* **then**
          Create a child rule-node of g, containing the rule r, with label c.
          for every non interpreted literal n in the antecedent of r,
            Create a child $n\theta$ for the rule-node with label $TDproj(r, \theta, c, n)$.
until no changes are made to T.
return T.
end build-tree.

Figure 3.3: Top down creation of the query-tree

Encoding Symbolic Derivations in the Query-tree

As stated, the query-tree *encodes* a set of symbolic derivations. Intuitively, a symbolic derivation is encoded in the query-tree if it can be constructed by choosing one rule-node for every goal-node. In doing so, we can expand an unexpanded goal-node with the children of its expanded equivalent. Formally, encoding is defined as follows:

Definition 3.1: A symbolic derivation d is encoded by the query-tree T if there exists a mapping ψ from the nodes of d, that do not have interpreted predicates, to the nodes of T that satisfies the following conditions:

E0. If $g_1, \ldots, g_n$ are the children goal-nodes of r in d, then $\psi(g_1), \ldots, \psi(g_n)$ are the children of $\psi(r)$ in T.

E1. For every rule-node $r \in d$, the rule in $\psi(r)$ is the same as the rule in r.

will not be part of a symbolic derivation, since the leaves of every symbolic derivation need to be of EDB predicates.

procedure shake-tree(T)
begin
/* Step 1: Marking reachability from the leaves */
Mark all EDB nodes in T as *accessible;*
repeat
if all children of a rule-node r are *accessible* **then** mark r as *accessible;*
if at least one child of a goal-node g is *accessible* **then** mark g as *accessible;*
if $Eq(\dot{g}) = g_1$ and g_1 is *accessible* **then** mark $\dot{g}$ as *accessible;*
until no new nodes are marked;

/* Step 2: Marking reachability from the root */
if g is a root of T and is *accessible* **then** mark it as *relevant;*
repeat
if g is a *relevant* goal-node, r is a child rule-node of g, and
all children of r are *accessible*
then mark r and its children as *relevant;*
if a goal-node g is *relevant* and either $g = Eq(g_1)$ or $\dot{g}_1 = Eq(g)$
then mark $\dot{g}_1$ as *relevant;*
until no new nodes are marked;
Remove all nodes that are not marked *relevant.*
/ If a there is a node g which is marked relevant, but its father rule-node is not marked relevant, then there must be some other node g_1 in that tree such that the father of $\dot{g}_1$ is marked relevant, and either $Eq(g_1) = g$ or $Eq(\dot{g}) = g_1$. Let θ be the isomorphism between g and $\dot{g}_1$ and let T_1 be the subtree of g. Make $T_1\theta$ the subtree of $\dot{g}_1$, and remove T_1 from the query-tree. */*
end shake-tree.

Figure 3.4: Shaking the query-tree

E2. The node $\psi(root(d))$ is a root in the query-tree.

E3. If r is a child of the goal-node $\dot{g}$ in d then:

1. If $\psi(\dot{g})$ is expanded in T, then $\psi(r)$ is a child of $\psi(\dot{g})$.
2. If $\psi(\dot{g})$ is not expanded in T, then $\psi(r)$ is a child of its expanded equivalent, $Eq(\psi(g))$.

Note that if d is encoded by the query-tree then for every goal-node $n \in d$, the node $\psi(n)$ is a variable renaming of n.

Finally, given a labeling scheme, we need to show that the symbolic derivation trees encoded by the query-tree are exactly those that satisfy Π. In doing so, we will be aided by the correspondence between the labels of the tree and the labels given by the finite labeling scheme. This correspondence is captured by the *label-preserving* property:

Definition 3.2: Let Π be a property of symbolic derivations that can be identified by a finite labeling scheme that assigns a label $L(n)$ to a node n in a symbolic derivation tree. Let T be a query-tree in which the label of a node n is denoted by $TL(n)$. The query-tree T is label-preserving w.r.t the labeling scheme L, if for any symbolic derivation d that is encoded by T, the equation $\phi(L(n)) = TL(\psi(n))$ holds, where ψ is the node-mapping from d to T, and ϕ is the variable renaming from n to $\psi(n)$. ∎

In words, the query-tree is label preserving if the mapping of the nodes also preserves the labels.

The Method: Summary

The general method for building a query-tree can be summarized by the following steps. To establish properties of a set of derivations $\mathcal{D}$, we do the following:

- Define a property Π of symbolic derivations, such that we can establish the desired properties of $\mathcal{D}$ by inspecting nodes in the symbolic derivations satisfying Π.
- Find a finite labeling scheme for Π.
- Describe a method for assigning labels to nodes in the query-tree.
- Show that the resulting query-tree encodes exactly the symbolic derivations that satisfy Π.

In the subsequent sections we describe several instances of this general method. Moreover, the general method provides a useful conceptual framework in which we can devise new labeling schemes for encoding sets of derivations.

Complexity

The time taken to build the query-tree (and therefore of deciding strong irrelevance) is dominated by the number of nodes in the tree. The other costs are those of checking whether two nodes are equivalent and of creating labels, both of which are polynomial in the size of a node. We observe that the number of internal nodes in the tree is

bounded by the number of possible non-isomorphic labels l, and therefore, the size of the tree can be at most lb, where b is the maximum number of literals in an antecedent of a rule. In the cases we consider, the number of labels depends *only* on the arity of predicates in the KB (and may be exponential in that number). It does not depend on the number of rules in the KB (and, of course, does not depend on the number of ground facts!). This is an important distinction because arities of predicates tend to be small (e.g., frame systems employ mostly binary predicates), and therefore, the algorithms will scale up to knowledge bases with many rules and ground facts.

3.2 Horn Rules With Interpreted Predicates

In this section, we consider the problem of building a query-tree that encodes the set of derivations of a query from a set of Horn rules $\mathcal{P}$ that may have interpreted predicates from a constraint language $\mathcal{L}$. Building such a query-tree will provide an algorithm for deciding strong irrelevance for the case where $\mathcal{D}_0 = \mathcal{D}_\psi$, i.e., deciding $SI(\phi, \psi, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_\psi)$.

Our first step is to define the set of symbolic derivations Π_{sat} that will be encoded by the query-tree. As explained earlier, these are the symbolic derivations in which the interpreted predicates on the variables are satisfiable. Formally, let d be a symbolic derivation that includes the rule-nodes $r_1, \ldots, r_m$, and let c_i be the conjunction of the literals with interpreted predicates that are children of r_i. Let

$$c_d = c_1 \wedge \ldots \wedge c_m.$$

The derivation d is a member of Π_{sat} if the constraint c_d is satisfiable.

The property in which we are interested is finding whether a ground atomic formula or a rule can appear in a derivation of the query. Inspecting the symbolic derivations in Π_{sat} is enough to verify this property:

Lemma 3.3:

1. *A ground formula $p(a_1, \ldots, a_n)$ can be part of a derivation of the query ψ if and only if there is some node $n = p(X_1, \ldots, X_n)$ in a symbolic derivation $d \in \Pi_{sat}$ such that $a_1, \ldots, a_n$ satisfies c_n, where c_n is the projection of c_d on the variables $X_1, \ldots, X_n$.*

2. *A rule r in the knowledge base can be part of a derivation of the query ψ if and only if some symbolic derivation $d \in \Pi_{sat}$ includes a rule-node containing the rule r.*

Proof: To prove Part 1, suppose there exists a symbolic derivation tree $d \in \Pi_{sat}$ and $a_1, \ldots, a_n$ satisfies the projection of c_d on the variables $X_1, \ldots, X_n$ of the node

$n \in d$. Therefore, there is some mapping θ of the variables of d to constants such that $X_i = a_i$ for $1 \leq i \leq n$ and such that applying θ to d will satisfy the constraints c_d. Applying θ to d will yield a derivation of v that uses $p(a_1, \ldots, a_n)$. Conversely, suppose there is a derivation d_0 of v that uses $p(a_1, \ldots, a_n)$. We can replace all the constants in d_0 by variables resulting in a symbolic derivation d. Clearly, the symbolic derivation is a member of Π_{sat} and $a_1, \ldots, a_n$ satisfies the projection of c_d onto the variables of the corresponding node in d.

Part 2 follows from the observation that a symbolic derivation will have exactly the same rules as its ground derivation instances. ∎

To encode Π_{sat}, our labeling scheme will be the following. Given a symbolic derivation tree d and a node n with variables $X_1, \ldots, X_m$, the *constraint-label* of n, denoted by $L_{sat}(n)$ will be the constraint denoting the projection of c_d onto the variables of $X_1, \ldots, X_m$. Note that since the constraint language satisfies the Closure property, the label $L_{sat}(n)$ can be expressed as a sentence in the constraint language $\mathcal{L}$. Intuitively, the label denotes the set of tuples that can appear in the node n in ground instances of the symbolic derivation d. To show that L_{sat} is a finite labeling scheme, we observe the following:

1. The Finiteness property implies that the number of possible labels (i.e., the number of non-isomorphic constraints) is finite.

2. The label of a node can be determined by its children or by its father. The label of a goal-node g is the projection of the label of its child rule-node onto the variables of g. The label of a rule-node is the conjunction of the labels of its children.

3. A symbolic derivation d is a member of Π if and only if all its labels are satisfiable.

In order to build the query-tree, we need a method for assigning labels to nodes in the tree. The difficulty in doing so is that the label of a node may depend on nodes that appear below it in the tree. Therefore, we cannot construct the tree and assign labels in one top-down phase, since the decision whether to expand a node depends on knowing its exact label. This problem will arise also in the labeling scheme we consider in Section 3.4. In what follows we describe a general method for solving this problem.

The solution is based on the following observation about computing the labels $L_{sat}(n)$ for nodes in a given symbolic derivation tree d. Given a tree d, we can compute its labels by a two phase process. In the first phase, we start with the leaves of d and compute labels based on propagating the interpreted constraints in a bottom-up fashion. In the second phase, we compute the labels by propagating the

interpreted constraints in a top-down fashion.[6] The procedure is summarized below. The labels computed in the bottom-up phase are denoted by c_b and the final labels are denoted by c_f.

for every goal-node $g \in d$, $c_0(g) = True$.
for every rule-node r, $c_0(r)$ = the conjunction of the interpreted children of r.
Traverse the rule-nodes of d in bottom-up fashion.
for each node r **do**:
/* g is the father of r and $g_1, \ldots, g_m$ are its children. */
$c_b(r) = c_0(r) \wedge c_0(g_1) \wedge \ldots \wedge c_0(g_m)$.
$c_b(g)$ = Projection of $c_b(r)$ on the variables of g.
/* *Note that $c_b(g)$ is the formula denoting the relation which is the projection of $R_{c_b(r)}$ on the variables in g.* */
$c_f(root(d)) = c_b(root(d))$.
Traverse the rule-nodes of d in a top-down fashion.
for each rule-node r **do**:
$c_f(r) = c_f(g) \wedge c_b(r)$.
For every $n \in g_1, \ldots, g_m$, $c_f(n)$ = Projection of $c_f(r)$ on the variables of n.

The following theorem shows that this procedure correctly computes the constraint labels of a symbolic derivation tree. The proof is given in Appendix A.

Theorem 3.4: *Let d be a symbolic derivation tree. For every node $n \in d$, $c_f(n) = L_{sat}(n)$.*

The importance of this theorem is that we can create the query-tree in a way that mimics the computation of the labels in the two phase process. Specifically, we show below that whenever the labels can be computed in a two phase process, it is enough to precede the top-down creation of the query-tree (by procedure **build-tree**) by a bottom-up computation phase. Informally, in the bottom-up phase we compute all the possible bottom-up labels for predicates in the KB, i.e., all the labels $c_b(n)$ that can appear in symbolic derivation trees. Based on these labels, we create a new set of *refined predicates*. For every label c that we compute for a predicate p, we create a new predicate p^c. We then create a set of *refined rules* $\mathcal{P}_1$ for the refined predicates by trying all the possible substitutions of the refined predicates in the rules of $\mathcal{P}$. The query-tree is then created in a top-down fashion using procedure **build-tree** and the rules $\mathcal{P}_1$. A rule $r' \in \mathcal{P}_1$ is a refinement of a rule $r \in \mathcal{P}$, if the predicate names in r' are refinements of the corresponding predicates in r. Note that r' and r have the same variable patterns.

[6] A bottom-up ordering of the rule nodes is any ordering for which a node n is traversed after all its descendents. A top-down traversal is the reverse order of a bottom-up traversal.

As an example of a bottom-up computation, consider the knowledge-base in Figure 3.2. The initial labels of the EDB predicates are the constraints that are given for them, i.e., $\{badPoint(X), 100 < X < 200\}$, $\{goodPoint(X), 150 < X < 170\}$, $\{step(X,Y), X < Y\}$ and $\{bigStep(X,Y), X < 100, Y > 200\}$. With the rules r_4 and r_5, we create the following labels for *link*: $\{link^1(X,Y), X < Y\}$ and $\{link^2(X,Y), X < 100, Y > 200\}$. With rules r_2 and r_3, we create $\{path^1(X,Y), X < Y\}$ and $\{path^2(X,Y), X < 100, Y > 200\}$. Finally, with $path^1$ we create the label $\{goodPath^1(X,Y), 100 < X < Y < 170, Y > 150\}$. Note that substituting $path^2$ in r_1 will yield the inconsistent label for *goodPath*, and therefore we do not perform that substitution. The refined rules that are created are:

$r_1 : badPoint(X) \wedge path^1(X,Y) \wedge goodPoint(Y) \Rightarrow goodPath^1(X,Y)$.
$r_2^1 : link^1(X,Y) \Rightarrow path^1(X,Y)$.
$r_2^2 : link^2(X,Y) \Rightarrow path^2(X,Y)$.
$r_3^1 : link^1(X,Z) \wedge path^1(Z,Y) \Rightarrow path^1(X,Y)$.
$r_3^2 : link^2(X,Z) \wedge path^1(Z,Y) \Rightarrow path^2(X,Y)$.
$r_3^3 : link^1(X,Z) \wedge path^2(Z,Y) \Rightarrow path^2(X,Y)$.
$r_4 : step(X,Y) \Rightarrow link^1(X,Y)$.
$r_5 : bigStep(X,Y) \Rightarrow link^2(X,Y)$.

Formally, in a two phase computation process we assume the existence of the following:

1. Initial labels for goal-nodes in a symbolic derivation tree, $c_0(n)$. We assume the initial label of a goal-node depends only on the predicate of the node.

2. A function $BULabel(r, (g_1, \ldots, g_m), (c_b(g_1), \ldots, c_b(g_m)))$ that accepts a rule-node, its subgoals and their respective bottom-up labels and computes the bottom-up label $c_b(r)$ for the rule-node.

3. A function $BUproj(r, \theta, c_b(r), g)$ that accepts a rule-node that contains a rule r, the unifier θ with which r was unified with its father goal-node g, and the bottom-up label of the rule-node, and returns a bottom-up label for its father goal-node g.

4. Functions $TDLabel$ and $TDProj$ as used in procedure **build-tree**, for computing the top-down labels.

We define the result of the two phase computation as follows (which is a generalization of the computation of L_{sat}):

- If g is a leaf in the tree, $c_b(g) = c_0(g)$.
- If r is a rule-node with children $g_1, \ldots, g_m$ and father g, then $c_b(r) = BULabel(r, (g_1, \ldots, g_m), (c_b(g_1), \ldots, c_b(g_m)))$, and $c_b(g) = BUproj(r, \theta, c_b(r), g)$.

- $c_f(root(d)) = c_h(root(d))$.
- $c_f(r) = TDLabel(r, c_f(g), g)$.
- $c_f(g_i) = TDproj(r, \theta, c_f(r), g_i)$.

Definition 3.5: The labeling scheme L is said to be 2-phase computable if $c_f(n) = L(n)$ for every node in every symbolic derivation tree. ∎

The bottom-up phase of building the query-tree is shown in Figure 3.5. All the labels created for a predicate p in this phase use the variables $X_1, \ldots, X_n$, where n is the arity of p. Therefore, we omit the second argument from $BUproj$, assuming it uses these standard variable names. The complete query-tree construction is described in Figure 3.6. After creating the query-tree we ignore the refinements of the predicates. That means that if n is a node in the tree of a predicate p^c and a label $c_f(n)$, we treat it as if it is a node of the predicate p with the same label. Note that the query-tree may actually be a forest of trees if the bottom-up phase computes more that one label for the query predicate.

procedure create-refined-rules($\mathcal{P}$)
begin
/* Constructing bottom-up labels P for every predicate p. */
  for every EDB predicate $p \in \mathcal{P}$, $P = \{c_0(p)\}$.
  for every IDB predicate $p \in \mathcal{P}$, $P = \{\}$.
  $\mathcal{P}_1 = \{\}$ /* $\mathcal{P}_1$ will be the set of refined rules */
  repeat
    Let r be the rule $q_1 \wedge \ldots \wedge q_m \Rightarrow h$.
    Let $c_i \in Q_i$, for $1 \leq i \leq m$.
    $c = BULabel(r, (g_1, \ldots, g_m), (c_1, \ldots, c_m))$.
    if c is consistent **then**
      $c_h = BUproj(r, c, h)$.
      Add c_h to H.
      Add the rule $q_1^{c_1} \wedge \ldots \wedge q_m^{c_m} \Rightarrow h^{c_h}$ to $\mathcal{P}_1$.
  until no new labels or rules are created.
end create-refined-rules

Figure 3.5: Creating the refined rules.

The following theorem states that whenever there is a 2-phase computable labeling scheme for the set of symbolic derivations Π, then the procedure **build-query-tree** will build a query-tree that encodes precisely the set of derivations Π.

procedure build-query-tree($\mathcal{P}$, q)
/* $\mathcal{P}$ is the set of rules. q is the query predicate. */
$\mathcal{P}_1$ =create-refined-rules($\mathcal{P}$);
for every label c of q **do**
T_c = **build-tree** ($\mathcal{P}_1$, q^c, c);
Query-tree = $\bigcup_{c \in Q}$ **shake-tree** (T_c);
end build-query-tree.

Figure 3.6: Building a query-tree

Theorem 3.6: *Let L be a 2-phase computable finite labeling scheme for the set of symbolic derivations Π. Let T be the query-tree generated by procedure* **build-query-tree**:

1. *If d is a symbolic derivation tree in Π, then d is encoded in T, and the encoding is label-preserving.*

2. *Let d_1 be a partial symbolic derivation tree encoded by the query-tree (i.e., some of the leaves have IDB predicates), then there is a symbolic derivation tree $d \in \Pi$, such that d_1 is a prefix of d and the encoding (limited to the nodes mapped to d_1) is label preserving.*

3. *A node n appears in a symbolic derivation tree in Π with label $L(n)$ if and only if there is some node in the query-tree with label $c_f(n)$ such that $L(n)$ is equivalent to $c_f(n)$.*

Proof: In the proof we assume that the query is of the form $q(\bar{X})$ where $\bar{X}$ is a set of distinct variables. Therefore, we refer to the query simply as the predicate q. Note that we can always transform the query into such a form.

We first prove Part 1. Let d be a symbolic derivation. A simple bottom-up induction on the nodes of d shows that the bottom-up labels of d were computed by **create-refined-rules**, specifically:

- For every goal-node $g \in d$, of the predicate p and bottom-up label $c = c_b(g)$, p^c is a predicate in $\mathcal{P}_1$.

- Suppose r is a rule-node in d containing the rule $q_1 \wedge \ldots \wedge q_m \Rightarrow p$. Suppose r's father is g and children are $g_1, \ldots, g_m$, then the following rule is in $\mathcal{P}_1$:

$$q_1^{c_b(g_1)} \wedge \ldots \wedge q_m^{c_b(g_m)} \Rightarrow p^{c_b(g)}.$$

Next, we show that d is encoded by the query-tree by mimicking the execution of procedure **build-tree**. We construct the encoding mappings ψ of the nodes as we go along.

We begin with the root of d and its child rule-node r. Let c denote $c_b(root(d))$. By the bottom-up construction we know that q^c is one of the refined predicates, and that there is a rule r_1 in $\mathcal{P}_1$ that is a refinement of the rule in r, and its antecedent is q^c. Therefore, procedure **build-tree** will be called with q^c. The procedure **build-tree** will begin with a node n with predicate q^c and the label c, and it will expand q^c with the rule r_1. Therefore, ψ will map $root(d)$ to n and will map r to r_1. The mapping ψ will map children of r to the respective children of $\psi(r)$. The bottom-up and top-down labels of $root(d)$ are the same, and therefore ψ is label preserving for $root(d)$. Since the labels specify completely the equality relations between the variables, $\psi(root(d))$ is a variable renaming of $root(d)$. Consequently, since r_1 is a refinement of the rule in r (and therefore they have the same variable patterns), the top-down labels of $\psi(r)$ and its children are determined by $\psi(root(d))$ (using the functions $TDLabel$ and $TDProj$) in the same way that the top-down labels of r and its children are determined from $root(d)$. Therefore, the mapping ψ is also label preserving for r and its children, and specifically, $\psi(n)$ is a variable renaming of n for n being r or one of its children.

Let $r_1, \ldots, r_n$ be a top-down ordering of the rule-nodes of d. We prove the claim by induction on the i^{th} rule-node. We assume by induction that we have built an encoding mapping ψ that satisfies the conditions of Definition 3.1 for all the rule-nodes $r_1, \ldots, r_{i-1}$, and their children goal-nodes. Note that for $i = 1$ this is exactly the base case discussed above. We prove that Part 1 holds for r_i and for its children. Furthermore, we assume by induction that if g is a goal-node in d, then $\psi(g)$ is actually a goal-node of $p^{c_b(g)}$, when the refinements of the goal-nodes in the query-tree are considered.[7] Note that this assumption holds for the root of d.

Let g be the father of r_i in d, and assume that g is a node of the predicate p. By the inductive assumption, $\psi(g)$ is a node of the predicate $p^{c_b(g)}$. Assume $\psi(g)$ was expanded in the query-tree. It would have been expanded with the rule

$$r' : q_1^{c_1} \wedge \ldots \wedge q_m^{c_m} \Rightarrow h^{c_b(g)}$$

where $c_1, \ldots, c_m$ are the bottom-up labels of the children of g, and r' is a refinement of the rule in r_i. Denote the resulting rule-node in the query-tree by r. The mapping ψ will map r_i to the node r and the subgoals of r_i to the subgoals of r (therefore satisfying condition E0 of Definition 3.1). Note that r' is a refinement of the rule in r_i, and we ignore the predicate refinements in the resulting query-tree, the rule in r_i and in $\psi(r_i)$ are the same (as required by condition E1). Furthermore, condition E3

[7] I.e., we consider the refined predicates in the rules $\mathcal{P}_1$.

is also satisfied by ψ. As in the base case, since r_i and $\psi(r_i)$ contain the same rule, and $\psi(g)$ is a variable renaming of g, the top-down labels of $\psi(r)$ and its children are determined by $\psi(g)$ (using the functions $TDLabel$ and $TDProj$) in the same way that the top-down labels of r and its children are determined from g. Therefore, the mapping ψ is also label preserving for r and its children, and specifically, $\psi(n)$ is a variable renaming of n, when n is either r or one of its children.

If $\psi(g)$ was not expanded in the query-tree, it would be because $Eq(\psi(g))$ is expanded. In that case, n would be a child of $Eq(\psi(g))$. In this case, E3 is still satisfied by the second clause in its definition. E0 and E1 hold as before. Since the label of $Eq(\psi(g))$ is isomorphic to the label of $\psi(g)$ (and in particular, $Eq(\psi(g))$ is a variable renaming of $\psi(g)$), the mapping ψ will be label preserving also for r_i and its children.

To complete the proof of Part 1 we must show that none of the nodes $\psi(n)$ for $n \in d$ were deleted from the query-tree in the shaking phase. However, a simple bottom-up induction on the nodes of d will show that all nodes $\psi(n)$ were marked *accessible*, and a top-down induction will show that they were all marked *relevant* and therefore not deleted.

We prove Part 2 in two parts. First, we show that every partial derivation encoded by the query-tree is a prefix of a complete symbolic derivation encoded by the tree. Next we show that every symbolic derivation encoded by the tree is a symbolic derivation in Π.

To prove the first part, we note that every goal-node in the query-tree is the root of some symbolic derivation (and the label of that node is the label of the root of the tree). In proof, if g is a node in the query-tree, then there is a sequence of nodes $n_1, \ldots, n_m$ such that $n_m = g$ and n_i was marked *accessible* because of some n_j for $j < i$. The node g is a head of a symbolic derivation consisting of $n_1, \ldots, n_m$. A simple induction on the reverse order of these nodes shows that they were all marked *relevant* and are therefore all in the query-tree.

Consequently, given a partial derivation tree d_1 encoded in the query-tree, we can complete every IDB leaf of d_1 with a symbolic derivation, thereby constructing a complete derivation.

To complete the proof of Part 2, let d' be a symbolic derivation encoded by the tree. Simply consider the symbolic derivation tree d with exactly the same structure (i.e., the same rules). Because the labeling is 2-phase computable, the labels of d' will be identical to the labels of d. Since the query-tree does not contain nodes with the *inconsistent* label, d' will be a member of Π.

Part 3 follows from the first two parts and the observation that every node in the query-tree appears in some partial derivation tree encoded by the query-tree. ∎

We complete this section with the following corollary that shows that the query-tree provides a sound and complete inference procedure for strong-irrelevance for

Horn-rule KBs with interpreted predicates.

Corollary 3.7: *Let $\mathcal{P}$ be a set of rules with interpreted predicates that satisfy the Closure, Equivalence, Satisfiability and Finiteness properties. Let T be the query-tree created for the rules $\mathcal{P}$ and the query q.*

1. *A formula $p(a_1, \ldots, a_n)$ is strongly irrelevant to q w.r.t. $\Sigma_{\mathcal{P}}$, (i.e., the irrelevance claim $SI(p(a_1, \ldots, a_n), q, \Sigma_{\mathcal{P}}, DI_i, \mathcal{D}_q)$ holds), if and only if there is no node n of p in T, such that $a_1, \ldots, a_n$ satisfies the constraint label of n, $c_f(n)$.*

2. *A rule r is strongly irrelevant to q if and only if it does not appear in T.*

Returning to the example in Figure 3.2, the rule r_5 is strongly irrelevant to *goodPath* because it does not appear in the query-tree. The atomic formulas of $step(X, Y)$ for which $X \leq 100$ or $Y \geq 170$ are also strongly irrelevant to the query.

3.2.1 Conjunctive Dense-order Constraints

One of the constraint languages which was covered by the discussion in the previous section is $\mathcal{L}^{\wedge,\vee}$, i.e., dense-order constraints with conjunction and disjunction. An important restricted language is that of dense-order constraints in which only conjunctions are allowed, which we denote by $\mathcal{L}^{\wedge}$. The atomic formulas of this language are of the form $(X\ \theta\ Y)$ or $(X\ \theta\ a)$, where X and Y are variables, a is a constant, and $\theta \in \{<, \leq, >, \geq, =, \neq\}$. Formulas in the language are either atomic or conjunctions of atomic formulas. In [Ullman, 1989], a complete polynomial-time decision procedure for this language is presented. Unfortunately, this language does not satisfy the Closure property we require. Specifically, given a sentence c in $\mathcal{L}^{\wedge}$, there is not always a sentence in $\mathcal{L}^{\wedge}$ that expresses the projection of c on a subset of its variables. The following is an example of such a case.

Example 3.8: Consider the conjunction:

$$X_1 \leq Z,\ Z \leq X_2,\ Z \neq X_3$$

This conjunction implies only one conjunctive constraint, $X_1 \leq X_2$, among the variables X_1, X_2, X_3. However, that does not fully describe all the constraints among X_1, X_2, and X_3. The constraint $X_3 \neq X_1 \vee X_3 \neq X_2$ is also implied by this conjunction, but since our language does not allow disjunctions, we cannot express this when trying to project the above conjunction onto X_1, X_2, and X_3. ∎

In creating a query-tree for constraints expressed in $\mathcal{L}^{\wedge}$, we modify the projection functions ($BUproj$ and $TDproj$). Since we cannot always express the exact projection of a constraint in $\mathcal{L}^{\wedge}$, these functions return a weaker constraint. Specifically, given a

constraint c on variables $\bar{X}$ and a subset $\bar{Y} \subseteq \bar{X}$, the functions return the conjunction of all the atomic constraints on variables in $\bar{Y}$ that are implied by c. In our example, the projection would be $X_1 \leq X_2$.

Consequently, the labels computed for nodes in the query-tree, which we denote by $c_f^{\wedge}(n)$, are weaker than the labels given by the labeling scheme L_{sat}. Therefore they do not describe the tightest constraint on every node in the query-tree. Fortunately, we can show that these labels are closely related to the labels given by L_{sat} and that we can use them to deduce strong irrelevance. Informally, the difference between the resulting labels $c_f^{\wedge}(n)$ and $L_{sat}(n)$ is that $c_f^{\wedge}(n)$ may be missing some disequalities ($\neq$) between the variables. Formally, we show the relation between them through the *least equality extension* of a conjunctive label, defined as follows:

Definition 3.9: Let c be a constraint on the variables $X_1, \ldots, X_n$ and constants $a_1, \ldots, a_m$. The least equality extension of c, denoted by $Max_{\neq}(c)$, is

$$c \wedge \{X_i \neq X_j \mid c \not\models X_i = X_j \wedge 1 \leq i, j \leq n\} \wedge \{X_i \neq a_j \mid C \not\models X_i = a_j \wedge 1 \leq j \leq m\}$$

■

The least equality extension simply adds disequalities between every pair of variables (or variable-and constant) that are not required to be equal by c. Note that the least equality extension is unique and therefore well defined since it can be built incrementally by examining each pair of variables (or variable and constant), and the order of the construction does not matter. In our example, the least equality extension of $X_1 \leq X_2$ is $\{X_1 \leq X_2 \wedge X_1 \neq X_3 \wedge X_2 \neq X_3\}$.

The following theorem relates $c_f^{\wedge}(n)$ to $c_f(n)$ (which was shown to be equivalent to $L_{sat}(n)$). It shows that the label $c_f^{\wedge}(n)$ is never stronger than $c_f(n)$ and that $c_f(n)$ is never stronger than $Max_{\neq}(c_f^{\wedge}(n))$. Recall that c_d denotes the conjunction of all the interpreted constraints on variables in a symbolic derivation d.

Theorem 3.10: *Let d be a symbolic derivation tree for which c_d is not necessarily satisfiable. For every node $n \in d$,*

1. $c_f(n) \models c_f^{\wedge}(n)$
2. *If $c_f(n)$ is satisfiable, then $Max_{\neq}(c_f^{\wedge}(n)) \models c_f(n)$.*
3. *If for all n, $c_f^{\wedge}(n)$ is satisfiable, then for all nodes n, $c_f(n)$ is satisfiable.*

The first and second parts guarantee the relationship between the labels in the query-tree constructed with $\mathcal{L}^{\wedge}$ and the labels of L_{sat}. The third part guarantees that when we build a query-tree with labels $c_f^{\wedge}$, the tree will not have any nodes that

shouldn't be included, i.e., all nodes are satisfiable (since they are weaker than the satisfiable constraint $Max_{\neq}(n)$). The proof of the theorem is given in Appendix A.

As stated, the labels in the resulting query-tree will not be the tightest ones possible. That means that if c is a label of a goal-node n in the query-tree, then actual tuples that can appear in valid derivations of the query may be a strict subset of those satisfying c. Formally, we can use the resulting query-tree to deduce irrelevance claims as follows:

Corollary 3.11: *Let $p(a_1,\ldots,a_n)$ be a ground atomic formula.*

1. *If $a_1,\ldots,a_n$ does not satisfy any of the constraint labels of nodes of p in the query-tree, then it is strongly irrelevant to the query.*
2. *Let g be a node of the predicate p in the query-tree. If $a_1,\ldots,a_n$ satisfies $Max_{\neq}(c_f(g))$, then $p(a_1,\ldots,a_n)$ is not strongly irrelevant to the query.*
3. *A rule r is in the query-tree if and only if r is not strongly irrelevant to the query.*

Proof: The query-tree encodes the set of symbolic derivations d in which for all $n \in d$ the labels $c_f^{\wedge}(n)$ are satisfied. Part 1 follows from Part 1 of Theorem 3.10 and Part 2 follows from Part 2 of that theorem. For Part 3, consider a symbolic derivation tree d encoded by the query-tree that uses a rule r. All of its labels are satisfiable, and therefore, by Part 3 of Theorem 3.10, c_d is also satisfiable. Consequently, there is a symbolic derivation of the query that satisfies Π_{sat} and includes r. Since the query-tree encodes a superset of the derivations in Π, then clearly if r does not appear in the query-tree, then it is strongly irrelevant to the query. ∎

Note that even though we do not get the tightest labels on the nodes when we use the language $\mathcal{L}^{\wedge}$, there may be advantages to using it over using $\mathcal{L}^{\wedge,\vee}$. Specifically, when we allow disjunctions, the constraints may become long (many disjuncts), and furthermore, checking equivalence of two constraints involving disjunctions is a more expensive operation. Therefore, the time to build the query-tree can be significantly affected.

The Number of Labels

As stated earlier, the time complexity of building the query-tree is dominated by the number of possible labels we can attach to nodes in the tree.

Consider the case of dense-order constraints expressed in $\mathcal{L}^{\wedge,\vee}$. In that case, every constraint label describes a set of possible orderings on the variables and constants in the rules. Given n variables and m constants, the number of possible total orderings

on them is exponential in $n+m$. Therefore, the number of constraint labels is doubly exponential in $n+m$. However, we note that it is sufficient to consider only total orderings on the variables and constants, and therefore the query-tree can be built in time that is singly exponential in $n+m$. However, in practice, the number of constraints that will be computed will be much smaller than the number of total orders and therefore, it is better not to limit ourselves to total orders. The number of labels expressible in $\mathcal{L}^{\wedge}$ is exponential in $n+m$, because it contains a subset of atomic formulas of which there is a polynomial number.

3.2.2 Rules With Function Symbols

When the set of rules contains function symbols, the Finiteness property may not hold. The source of the problem is that when a goal-node is unified with the head of a rule, new terms may be created, and therefore the number of labels that can be created may be infinite. Consider the following example.

Example 3.12: The following rules define the set of integers:

$s_1 : (X = 0) \Rightarrow integer(X)$
$s_2 : integer(X) \Rightarrow integer(X + 1)$

As shown in Figure 3.7(a), a top-down expansion of the tree for these rules will result in an infinite number of labels $\{Z_i = X - i\}$ for every integer, i. Therefore, the construction of the query-tree will not terminate. ∎

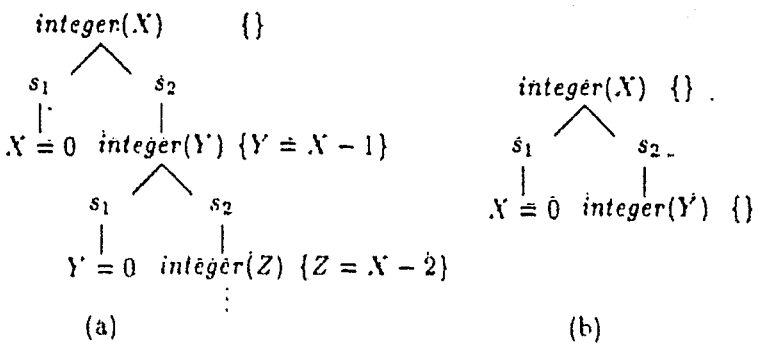

Figure 3.7: Query-tree with function symbols.

To build a query-tree in this case we can assign the nodes in the query-tree one of a finite set of labels $\mathcal{C}$. When we project a constraint on a subset of its variables, we

proceed by the following strategy. Given a constraint c and a subset of its variables $\bar{X}$, if there is no label in $\mathcal{C}$ which describes the exact projection $c|_{\bar{X}}$, we assign a member c_1 of $\mathcal{C}$ such that $c|_{\bar{X}} \models c_1$ and such that there is no other constraint $c_2 \in \mathcal{C}$ such that $c_2 \models c_1$ and $c|_{\bar{X}} \models c_2$. The constraint c_1 can be viewed as the best *approximation* to $c|_{\bar{X}}$ out of the finite number of labels $\mathcal{C}$. Consequently, the resulting labels in the query-tree are weaker than the tightest ones possible, and therefore, the query-tree provides only a sufficient condition for strong irrelevance. That means that a ground atomic formula which does not match any of the nodes in the tree is strongly irrelevant, but not vice versa.

One way to assign such a finite set of labels is to not allow new terms to be created in the labels (or to allow a maximum of k new terms, where k is fixed). For instance, in our example, if we do not allow new terms, we get the query-tree shown in Figure 3.7(b).

Finally, it should be noted that the problem with function symbols arises only when the rules are recursive. If they are not, then the number of labels will necessarily be finite (because the number of unifications is finite, and each unification may introduce only a finite number of new terms). Consequently, in such cases, the query-tree still provides a complete inference procedure for strong irrelevance (assuming the constraint language satisfies the properties described in the previous section).

3.3 Encoding Minimal Derivations

In this section, we consider another instance of the query-tree algorithm in which the query-tree is built to encode only the minimal derivations of the query. The definition of minimality that we consider (M1 from Chapter 2) states that a derivation is minimal if there are no two identical nodes, n_1 and n_2, such that n_1 is an ancestor of n_2. In Chapter 2 we showed that strong irrelevance w.r.t. this definition is equivalent to strong irrelevance w.r.t. the stronger definition $M2$, and provides a sufficient condition for strong irrelevance under the condition $M3$.

Example 3.13: Consider the following knowledge base, where e is the EDB predicate and p and p_1 are IDB predicates.

$r_1 : p(Y, X) \Rightarrow p(X, Y)$.
$r_2 : e(X, Y) \Rightarrow p(X, Y)$.
$r_3 : p(X, X) \Rightarrow p_1(X)$.

The rule r_1 can only appear in non-minimal derivations of p_1. Note however, that r_1 will appear in minimal derivations of p. ∎

As before, our first step is to find a set of symbolic derivations that the query-tree will encode. We build a query-tree that encodes the following set of symbolic derivations Π_{min}. A derivation d is a member of Π_{min} if:

1. $d \in \Pi_{sat}$ and
2. There is no pair of nodes $p(X_1, \ldots, X_n)$ and $p(Y_1, \ldots, Y_n)$ for any predicate p, such that $c_d \models (X_1 = Y_1) \wedge \ldots \wedge (X_n = Y_n)$, and such that $p(X_1, \ldots, X_n)$ is an ancestor of $p(Y_1, \ldots, Y_n)$. (Note that in this case, the two nodes in the tree would be identical). If two such nodes exist we would say that the tree contains a loop.

In showing the properties of Π_{min}, we make the following additional assumptions about the constraint language used in the rules:

Density: Suppose that $R_f(X_1, \ldots, X_n)$ is the relation consisting of the tuples satisfying the formula f, and for all $1 \leq i < j \leq n$, formula f does not imply $X_i = X_j$ or $X_i = a$, where a is a constant. Let $X^1, \ldots, X^k$ be k columns in the relation R_f, and let R' be the set of all tuples in R_f in which column i, $1 \leq i \leq k$ has the constant a_i, where $a_1, \ldots, a_k$ are arbitrary constants. The Density Property requires that R' be an infinite set or an empty set.

Intuitively, the property guarantees that if we are given a partial assignment to variables that satisfy a certain constraint, then we can complete the assignment in arbitrarily many ways. The reason the assumption is needed is that we want to guarantee that if a symbolic derivation tree d is in Π_{min}, then we can always find a corresponding ground derivation in which every variable in d is mapped to a distinct constant, and will therefore be a minimal derivation.

Equality connectivity: Suppose the variable X appears in the goal-nodes g_1 and g_2 in a symbolic derivation tree d. Let g be the least common ancestor goal-node of g_1 and g_2 in d. Then X appears in every goal-node on the path from g to g_1 and on the path from g to g_2.

Note that both of these assumptions hold for constraint languages using order predicates, as long as the domain of the variables is assumed to be dense, or for the constraint language containing only equality. When the rules do not contain interpreted predicates, they can be viewed as using the constraint language of equalities,[8] and therefore the Density property is satisfied. Furthermore, it should be noted that if these properties are not satisfied, then the query-tre still provides a sound inference procedure for strong irrelevance. This means that a node may appear in the query-tree and still be strongly-irrelevant to the query. It should be noted that even when these assumptions do not hold, finding examples in which the query-tree is not complete requires careful crafting of the rules.

[8] Because equalities can be represented implicitly by multiple occurrences of the same variable.

Under these assumptions, we can deduce properties of minimal derivations by inspecting symbolic derivations in Π_{min} as follows:

Lemma 3.14:

1. *If $p(a_1, \ldots, a_n)$ does not satisfy any of the labels of nodes of p in any tree $d \in \Pi_{min}$, then $p(a_1, \ldots, a_n)$ does not appear in any minimal derivation of the query.*

2. *If $p(a_1, \ldots, a_n)$ satisfies the constraint-label of some node n (of the predicate p) in a symbolic derivation tree $d \in \Pi_{min}$, and the equality relations between the constants $a_1, \ldots, a_n$ are only those that are entailed by the constraint label, then there is some minimal derivation of the query that uses $p(a_1, \ldots, a_n)$.*

3. *A rule r is used in a minimal derivation of the query if and only if r appears in some symbolic derivation $d \in \Pi_{min}$.*

Proof: To prove Part 1, if $p(a_1, \ldots, a_n)$ does not match any node in symbolic derivations in Π_{min}, there are two possibilities. The first is that $p(a_1, \ldots, a_n)$ does not match any node in symbolic derivations in Π_{sat}. If this is the case, clearly $p(a_1, \ldots, a_n)$ does not appear in any derivation of the query (by Lemma 3.3). The other possibility is that it only appears in derivations of Π_{sat} whose corresponding symbolic derivations contain a loop. However, every instance of a symbolic derivation that contains a loop will contain a loop, and will therefore not be minimal.

Part 3 is proved as follows. If $d \in \Pi_{min}$ uses the rule r, then $d \in \Pi_{sat}$. Therefore c_d is satisfiable. Consider an assignment ψ of the variables in d that satisfies c_d and such that ψ assigns two variables X_1 and X_2 the same value only if $c_d \models X_1 = X_2$. Since d contains no loops, the derivation $d\psi$ will be a minimal derivation. If it were not, that would imply that there are two nodes $g_1(\bar{X})$ and $g_1(\bar{Y})$ such that $c_d \models \bar{X} = \bar{Y}$.

Conversely, if there is a minimal derivation that uses r, consider its corresponding symbolic derivation d. Clearly, $d \in \Pi_{min}$.

The proof of Part 2 requires the Density property. Let $X_1, \ldots, X_m$ be the variables in d and assume $X_1, \ldots, X_n$ are the variables that appear in the node whose label is satisfied by $p(a_1, \ldots, a_n)$. We show that there is an assignment ψ to the variables of d that satisfies c_d and such that:

1. for $1 \leq i \leq n$, $X_i = a_i$ and

2. for $1 \leq i, j \leq m$, $X_i = X_j$ (or $X_i = a$) only if it is implied by c_d.

Applying ψ to d will yield a minimal derivation of the query that uses $p(a_1, \ldots, a_n)$.

We start by assigning $X_i = a_i$ for $1 \leq i \leq n$. Note that this mapping satisfies the second condition because $a_1, \ldots, a_n$ only satisfies the equalities required by the label it matches. We proceed by induction on i. Given assignments to $X_1, \ldots, X_i$ we need

to assign a value to X_{i+1}. If $c_d \models X_{i+1} = X_j$ for $j \leq i$, then we assign X_{i+1} the value assigned to X_j (similarly, if $c_d \models X_{i+1} = a$, we assign a).

We now show that there are an infinite number of assignments a_{i+1} to X_{i+1} such that $a_1, \ldots, a_{i+1}$ will satisfy c_d, and therefore we can choose a value a_{i+1} that is not assigned to any of $X_1, \ldots, X_i$.

Let $b_{i+2}, \ldots, b_m$ be an assigment to the variables $X_{i+2}, \ldots, X_m$ such that the tuple consisting of $\psi(X_1), \ldots, \psi(X_i), b_{i+2}, \ldots, b_m$ satisfies c_d. Note that $b_{i+2}, \ldots, b_m$ must exist because $\psi(X_1), \ldots, \psi(X_i)$ satisfies c_d. Now consider the selection on c_d in which $X_j = \psi(X_j)$ for $1 \leq j \leq i$ and $X_j = b_j$ for $i+1 \leq j \leq m$. This is the subset of R_d that is equal on the columns $X_1, \ldots, X_i, X_{i+2}, \ldots, X_m$, and therefore, by the Density property must be infinite. This means that there are an infinite number of values that X_{i+1} can take that will be consistent with $\psi(X_1), \ldots, \psi(X_i)$ and with c_d. ∎

It is important to note that Lemma 3.14 implies that if we can build a query-tree to encode Π_{min}, then strong irrelevance under minimal derivations is decidable. The only subtle point that needs to be considered is when we have atoms of the form $p(a_1, \ldots, a_n)$ that satisfy some label in some derivation tree in Π_{min}, but where $a_1, \ldots, a_n$ satisfy additional equalities not implied by the label. In such a case we can create a specialized predicate p' that enforces these equalities. For example, if we had an atom $p(X, X)$, we would create a predicate $p'(X)$ defined as $p(X, Y) \wedge (X = Y)$. We then consider every rule in the KB in turn. Whenever a rule can use the predicate p, we make another version of it that uses p'.[9] We then build a query-tree for the KB that includes the rules with p' and check if $p'(a'_1, \ldots, a'_m)$ matches a node in that query-tree (where $a'_1, \ldots, a'_m$ is the result of removing duplicate constants from $a_1, \ldots, a_n$).

Our next step is to devise a labeling scheme for Π_{min}. A label of a node n, denoted by $L_{min}(n)$ will be a pair (c, t) where $c = L_{sat}(n)$ and t will be the *tag* of n, defined as follows. We denote by $V(g)$ the variables that appear in the node g.

Definition 3.15: Let g be a goal-node in a symbolic derivation tree. Let S be the set of its ancestor goal-nodes that have only variables from $V(g)$ or constants. If S contains a node that is identical to g, the tag of g is *inconsistent*. Otherwise, the tag of g, denoted by $T(g)$ is $S \cup \{g\}$. The tag of a rule-node $r \in d$ is the tag of its father goal-node. ∎

Two labels (c_1, t_1) and (c_2, t_2) are the same if there is an isomorphism between c_1 and c_2 which is also an isomorphism between t_1 and t_2.

Lemma 3.16: *The labeling L_{min} is a finite labeling scheme for Π_{min}. There exist functions BU'_{min} and TD_{min} such that L_{min} is 2 phase computable.*

[9] If p appears in two or more subgoals of a rule r we make a version for every subgoal.

Proof: To show that there is a finite number of labels, it suffices to show that there is only a finite number of possible tags. Consider the atoms of a predicate p in a tag. The number of different atoms of p is the number of possible variable patterns of the arguments of p, which is the number of ways to partition the arguments of p into equivalence classes. This number is exponential in the arity of p (cf. [Graham *et al.*, 1989], pg. 244). Therefore, the number of atoms that may appear in a tag is exponential in the maximum arity of predicates in $\mathcal{P}$. Consequently, since a tag is a set of atoms, the number of possible tags is doubly exponential in the arity.

Next, we observe that the tag of a goal-node can be determined by the tag of its father. Let g be a goal-node whose grandfather goal-node is g_1, and suppose $g' \in T(g)$, i.e., g' is an ancestor of g that has only variables from $V(g)$ or constants. By the Equality connectivity assumption, $V(g') \subseteq V(g_1)$ (because g_1 is on the path from g' to g). Therefore $g' \in T(g_1)$, and g' will be in $T(r)$, where r is the father rule-node of g. Consequently, $T(g)$ can be determined by inspecting only the atoms in $T(r)$.

Since $T(g)$ can be determined by the tag of its father, it also follows the L_{min} is 2 phase computable. In the first phase, the computation is identical to that of L_{sat}, and so we define BU'_{min} to be identical to $BULabel$. In the second phase, the function $TD_{min}(r, \theta, (c, T(r)), g)$ will compute a label $(c', T(g))$ as follows. The first component is simply that computed by $TDLabel$, i.e., $c' = TDLabel(r, \theta, c, g)$. To compute the tag of g, simply inspect the atoms in the tag of its father r. Aside from g itself, any atom that includes only variables from g and constants will be in $T(g)$. If $g \in T(r)$ then $T(g)$ will be *inconsistent*.

Finally, we need to show that Π_{min} can be verified by inspecting the labels L_{min} of the nodes in the tree d. This follows from the following observation: $d \in \Pi_{min}$ if and only if for all $n \in d$, $L_{min}(n)$ is not inconsistent, i.e., if $L_{min} = (c, t)$, then c is satisfiable and t is not *inconsistent*.

In proof, suppose there are two identical nodes n_1 and n_2 in d such that n_1 is an ancestor of n_2, and let r be the father of n_2. The tag $T(r)$ would contain n_1 (because of the Equality connectivity assumption) and therefore, when computing $T(n_2)$ we would get an inconsistent label. Conversely, if for some node n $T(n)$ is inconsistent, it must be the case that $n \in T(r)$, where r is the father of n. This means that one of the ancestors of n is identical to n. ∎

The corollary below follows from Lemma 3.16 and Theorem 3.6.

Corollary 3.17: *The procedure* **build-query-tree** *with functions BU'_{min} and TD_{min} will compute a query-tree that encodes precisely the symbolic derivations in Π_{min}.*

The following example illustrates the use of tags and shows that they are indeed necessary in order to derive strong irrelevance under minimal derivations.

Example 3.18: Consider the following knowledge base. The e_i's are the EDB predicates.

$r_1 : q(X, Z) \wedge e(Z, Y) \Rightarrow q(X, Z)$
$r_2 : e_1(X, Y) \Rightarrow q(X, Y)$
$r_3 : p(X, Y) \Rightarrow q(X, Y)$
$r_4 : e_2(X, Y) \Rightarrow p(X, Y)$
$r_5 : q(X, Y) \Rightarrow p(X, X)$

Since there are no interpreted constraints in the rules and no equalities between variables, the constraint labels of all the nodes in the query-tree (see Figure 3.8(a)) will have the $True$ constraint. Note, that we do not expand the node $q(X, Y)$ with rule r_3 because it will result in a subgoal $p(X, Y)$ which is identical to the root of the tree, and will therefore produce an inconsistent tag. However, we can expand the node $q(X, Z)$ with r_3 without creating a loop. Therefore it is important to distinguish between $q(X, Y)$ and $q(X, Z)$ even though they have the same constraint label L_{sat}. Since $L_{min}(q(X, Z))$ is the same as $L_{min}(q(X, Y))$, we do not expand the latter node.

Figure 3.8(b) shows the query-tree that would be obtained using the labeling L_{sat}. In this case, the node $q(X, Y)$ would have been expanded with r_3 and therefore, the query-tree would encode also non-minimal derivations. ∎

3.4 Rules with Negations in the Antecedents

Recall from Chapter 2 that if we have a set of rules with stratified negation, then strong irrelevance is undecidable (Lemma 2.18). In this section we discuss a restricted case of stratified rules in which only literals of EDB predicates may appear negated in the rules. In this case, a derivation can be viewed as a tree as before, except that some of the leaves of the tree may be negated literals. A negated literal $\neg e(\bar{a})$ is considered to be satisfied if the ground atom $e(\bar{a})$ is not in the knowledge base.

As before, the query-tree will encode a set of symbolic derivations, in this case denoted by Π_{neg}. A symbolic derivation d will be a member of Π_{neg} if

1. $d \in \Pi_{sat}$ and
2. There is no pair of ground atoms $e(\bar{X})$ and $\neg e(\bar{Y})$ such that $c_d \models (\bar{X} = \bar{Y})$, i.e., there is no pair of contradictory leaf atoms.

The second condition guarantees that the symbolic derivation is satisfiable, i.e., it does not require that both an atom and its negation be in the knowledge base.

The following theorem shows that encoding Π_{neg} will enable us to decide strong irrelevance:

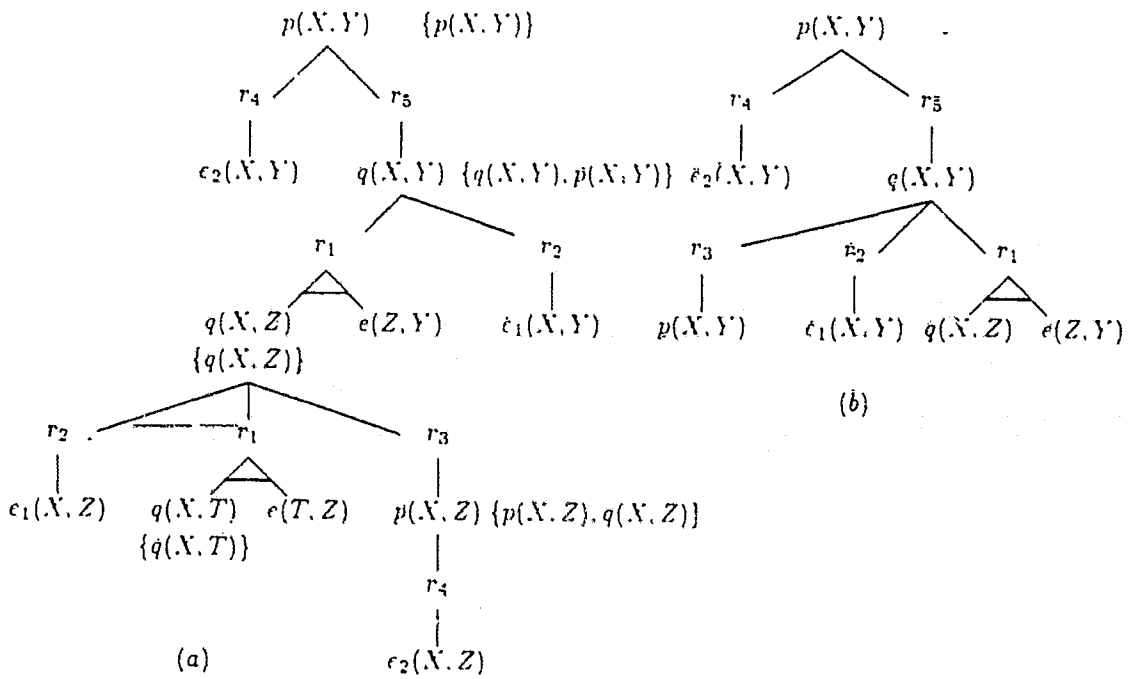

Figure 3.8: (a) A query-tree with node-tags (shown only for IDB goal-nodes). (b) The query-tree that would have been produced without considering tags.

Lemma 3.19:

1. *If $p(a_1, \ldots, a_n)$ does not satisfy any of the labels of nodes of p in any tree $d \in \Pi_{neg}$, then $p(a_1, \ldots, a_n)$ does not appear in any valid derivation of the query.*

2. *If $p(a_1, \ldots, a_n)$ satisfies the constraint-label of some node n (of the predicate p) in a symbolic derivation tree $d \in \Pi_{neg}$, and $a_1, \ldots, a_n$ satisfies only the equalities required by the constraint label, then there is some database in which $p(a_1, \ldots, a_n)$ is used in a derivation of the query.*

3. *A rule appears in a symbolic derivation in Π_{neg} if and only if it is not strongly irrelevant to the query.*

Proof: The proof is very similar to that of Lemma 3.14. For Part 1, suppose $p(a_1, \ldots, a_n)$ was part of a valid derivation d' of the query, and let d be a symbolic derivation corresponding to d'. Clearly, $d' \in \Pi_{neg}$.

The proof of Part 2 follows from the claim shown in the proof of Lemma 3.14. There we proved the following claim. Let $X_1, \ldots, X_m$ be the variables in d and assume $X_1, \ldots, X_n$ are the variables that appear in the node whose label is satisfied by $p(a_1, \ldots, a_n)$. Then there is an assignment ψ to the variables of d that satisfies c_d and such that:

1. for $1 \leq i \leq n$, $\psi(X_i) = a_i$ and
2. for $1 \leq i, j \leq m$, $\psi(X_i) = \psi(X_j)$ (or $\psi(X_i) = a$) only if the equality is implied by c_d.

Suppose we apply ψ to d. Since $d \in \Pi_{neg}$, d does not have two contradicting literals. Therefore, $d\psi$ will have two contradicting literals only if two distinct variables X_i and X_j, such that $c_d \not\models X_i = X_j$, were mapped by ψ to the same constant. However, that contradicts the assumption on ψ. Therefore, $d\psi$ is a valid derivation of the query that uses $p(a_1, \ldots, a_n)$.

Part 3 is proved exactly as in the proof of Lemma 3.14. ∎

As in the case of minimal-derivations, it should be noted that if we can build a query-tree to encode precisely Π_{neg}, then strong irrelevance is decidable for such rules. The next step is to devise a labeling scheme for Π_{neg}. For clarity, we begin with the case in which there are no interpreted predicates in the rules. Furthermore, we assume that no positive subgoal or head of a rule in the KB has the same variable in two or more columns. Rules that do not satisfy this assumption can be converted into such a form using equality constraints and will therefore be covered later.

Note that the Density property holds trivially in this case. Furthermore, all unifications of rules with subgoals are trivial. Our labeling scheme in this case will be L_{neg}, which is defined as follows. A label of a node n is a pair (c, e), where $c = L_{sat}(n)$ and e is the *EDB-label* of n, defined as follows:

Definition 3.20: EDB-label: Let r be a rule-node in a symbolic derivation tree d and let g be its father goal-node. Let S be the set of all EDB literals that appear in the subtree rooted in r. We say that the set S is *consistent* if it does not contain an atom A and its negation $\neg A$. If S is consistent, then the EDB-label of rule-node r $e(r)$ is the set of literals of S that contain only variables from g or constants. If S is not consistent, then the EDB label of r is the inconsistent label. The EDB-label of an IDB goal-node g is the same as the EDB label of its child rule-node. The EDB-label of an EDB goal-node is the set containing itself. ∎

As before, two sets of literals e_1 and e_2 are considered to be identical EDB-labels if there is 1-1 mapping ψ of the variables of e_1 to the variables of e_2 such that $\psi(e_1) = e_2$.

Example 3.21: Consider the following knowledge base:

$T1 : e(X, Y) \Rightarrow p(X, Y)$
$T2 : e(X, Z) \wedge \neg g(X) \wedge \neg g(Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$
$T3 : p(X, Y) \wedge g(Y) \wedge \neg e(X, Y) \Rightarrow c(X, Y)$

Rules $T1$ and $T2$ define a p (path) relation in terms of EDBs e (edge) and g (good nodes). Rule $T3$ defines a c (connectivity) relation. Figure 3.9 show a symbolic derivation tree created from this knowledge base with its EDB labels. ∎

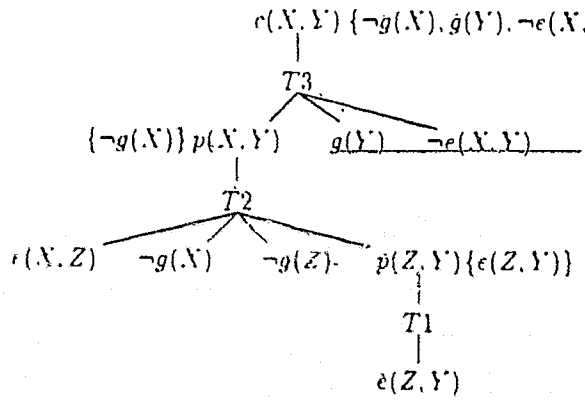

Figure 3.9: Symbolic derivation tree with EDB labels

The following proposition shows that an EDB-label can be computed from the EDB-labels of its subgoals.

Proposition 3.22: *The EDB-label of a rule-node r can be computed by the EDB-labels $e_1, \ldots, e_m$ of its subgoals as follows. Let S be $e_1 \cup \ldots \cup e_m$. If S is consistent, then the EDB-label of r is the set of literals in S that contain only variables appearing in r's father, g. Otherwise, the EDB-label of r is inconsistent.*

Proof: The proof follows from the Equality Connectivity assumption. Specifically, if a variable X appears in two nodes n_1 and n_2 in a symbolic derivation tree such that n_1 is an ancestor of n_2, then it appears in every goal-node on the path from n_1 to n_2. Therefore, if an EDB-literal g_e contains only variables that appear in one of its ancestors g, then either g_e is a subgoal of g or there is a subgoal g_1 of g which is an ancestor of g_e. In the latter case, g_e will be in the EDB-label of g_1 because all the variables of g_e must appear in g_1. In both cases, g_e will be in the set S defined above. ∎

Lemma 3.23:

1. *The number of EDB-labels is finite.*
2. *A symbolic derivation tree d is a member of Π_{neg} if and only if none of the rule-nodes of d has the inconsistent label.*
3. *There exist functions $BUneg$ and $TDneg$, such that L_{neg} is 2 phase computable.*

Proof: To prove Part 1, we observe as in the case of node tags, that the number of atoms that can appear in an EDB label is exponential in the maximum arity of predicates in $\mathcal{P}$ (though in this case, the number is double since they can either appear positively or negatively). Since an EDB-label is a set of atoms, the number of EDB-labels is doubly exponential in the arity of the predicates.

Part 2: Suppose d has a node r with an inconsistent label (note that the inconsistency can only come from the EDB-label, since equality constraints are always satisfiable). That means that r has an atom and its negation in its subtree, and therefore, $d \notin \Pi_{neg}$. Conversely, suppose d contains an atom A and its negation $\neg A$. Let r be the least common ancestor goal-node of A and $\neg A$. The EDB-label of r will be inconsistent.

To prove Part 3, we define two functions *BUneg* and *TDneg*. Note that under the assumptions we have made, the constraint part of L_{neg} is always the *True* constraint. The EDB-label part is computed by *BUneg* as defined in Proposition 3.22. The function *TDneg* is simply the identity function, since the EDB-label does not change in the top-down phase. The proof follows from Proposition 3.22. ∎

Corollary 3.24: *The procedure* **build-query-tree** *with the functions BUneg and TDneg will compute a query-tree that encodes precisely the set of derivations Π_{neg}.*

Returning to Example 3.21, the first step of the query-tree algorithm will produce the following EDB labels. Note that to avoid confusion, we use variables in the EDB-labels that are disjoint from those that appear in the tree. Rule $T1$ derives the EDB label $\{e(X_1, X_2)\}$ for p. Using the EDB label $\{e(X_1, X_2)\}$ for p, rule $T2$ derives the EDB label $\{\neg g(X_1)\}$ for p. Using the EDB label $\{\neg g(X_1)\}$ for p in rule $T2$ generates the isomorphic label $\{\neg g(X_1)\}$ for p. Thus, no more EDB labels for p can be derived. Using the EDB label $\{\neg g(X_1)\}$ for p, rule $T3$ derives the EDB label $\{\neg g(X_1), g(X_2), \neg e(X_1, X_2)\}$ for c. Using the EDB label $\{e(X_1, X_2)\}$ for p in rule $T3$ generates an inconsistency, and no new EDB label is derived. Consequently the set of refined rules is the following:

$T1' : e(X, Y) \Rightarrow p^{\{e(X_1, X_2)\}}(X, Y)$
$T2'_a : e(X, Y) \wedge \neg g(X) \wedge \neg g(Z) \wedge p^{\{e(X_1, X_2)\}}(Z, Y) \Rightarrow p^{\{\neg g(X_1)\}}(X, Y)$

$T2'_b : e(X,Y) \wedge \neg g(X) \wedge \neg g(Z) \wedge p^{\{\neg g(X_1)\}}(Z,Y) \Rightarrow p^{\{\neg g(X_1)\}}(X,Y)$
$T3' : p^{\{\neg g(X_1)\}}(X,Y) \wedge g(Y) \wedge \neg e(X,Y) \Rightarrow e^{\{\neg g(X_1), g(X_2), \neg e(X_1,X_2)\}}(X,Y)$

The query-tree created for this example is shown in Figure 3.10. Note that we do not expand the rightmost node $p^{\{\neg g(X_1)\}}(Z,Y)$, since its EDB label is the same as the EDB-label of node $p^{\{\neg g(X_1)\}}(X,Y)$.

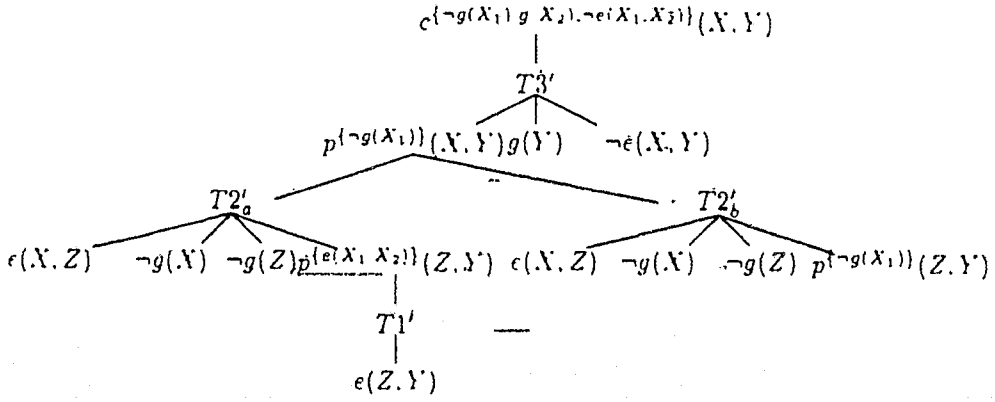

Figure 3.10: The query-tree built for the program P_1.

Adding Interpreted Predicates

In the previous section we showed that we can compute L_{neg} in a 2 phase procedure. However, that result depended on the observation that we knew all the equality relations between variables in the tree during the bottom-up phase. Specifically, if a variable in the body of a rule must be equal to a variable in the head, then we would know that in the bottom-up computation of $BUneg$. The assumptions we made in the previous section guaranteed that property because rules could not imply any equality relations between variables. However, when we allow the rules to have interpreted literals, this assumption may not hold, and therefore, the EDB-label computed may not be correct. The following example illustrates the problem.

Example 3.25: Consider the following rules:

$r_1 : e_1(X,Y) \wedge e_2(Z) \wedge X \leq Z \leq Y \Rightarrow p(X,Y)$
$r_2 : e_3(X,Y,T) \wedge \neg e_2(T) \wedge X \geq T \geq Y \Rightarrow q(X,Y,T)$
$r_3 : p(X,Y) \wedge q(X,Y,T) \Rightarrow s(X,Y)$

The EDB-labels would be computed as follows. Using r_1, we first create a label $\{e_1(X_1,X_2)\}$ for p. Note that $e_2(Z)$ is not included in the EDB-label because Z does not appear in the head of r_1 nor is it known to be equal to one of the variables in it.

Next, with r_2 we will create a label $\{e_3(X_1, X_2, X_3), \neg e_2(X_3)\}$ for q. Finally, with r_3 we will create a label $\{e_1(X_1, X_2)\}$ for s. However, considering the constraint label of $s(X, Y)$ implies that $X = Y = T$, and therefore $X = Y = Z$. Consequently, the symbolic derivation of s is inconsistent because it contains both $e_2(Z)$ and $\neg e_2(T)$. However, the EDB-labels computed were all consistent and so we were not able to detect the contradiction. ∎

Fortunately, there is an easy fix for this problem. Recall that after computing the constraint labels L_{sat} of nodes in a symbolic derivation tree, the labels are as restrictive as possible, and therefore describe all the equality constraints between variables. Thus, we can create a new set of adorned predicates and rules from the query-tree that have the constraints completely propagated. Specifically, if g is a goal-node of the predicate p in the query-tree (built with *BULabel* and *TDLabel*), and $L_{sat}(g) = c$, then we create an adorned predicate p^c. If r is a rule-node in the query-tree, and we created an adorned predicate p^c from its father and predicates $q_1^{c_1}, \ldots, q_m^{c_m}$ from its children, then we create the adorned rule

$$q_1^{c_1} \wedge \ldots \wedge q_m^{c_m} \wedge L_{sat}(r) \Rightarrow p^c$$

in which none of the positive literals in the antecedent have the same variable in different columns. We denote the new set of rules by $\mathcal{P}_1$. We note that the rules $\mathcal{P}_1$ are equivalent to $\mathcal{P}$ w.r.t the query q. This means that $(\mathcal{P} \cup D) \vdash q(\bar{a})$ if and only if there is some c such that $(\mathcal{P}_1 \cup D) \vdash q^c(\bar{a})$. Moreover, when we build a query-tree for $\mathcal{P}_1$ then the constraint labels are completely known in the bottom-up phase, and therefore, we can compute the EDB-labels in parallel with the constraint labels. Returning to our example, we would create the following rules from the query-tree (note that every predicate has only one adornment, so we do not change the predicate names):

$r_1 : e_1(X, Y) \wedge e_2(Z) \wedge X = Z = Y \Rightarrow p(X, Y)$
$r_2 : e_3(X, Y, T) \wedge \neg e_2(T) \wedge X = T = Y \Rightarrow q(X, Y, T)$
$r_3 : X = Y = T \wedge p(X, Y) \wedge q(X, Y, T) \Rightarrow s(X, Y)$

Computing EDB-labels with these rules will result in an inconsistent label for s.

3.5 Complexity

As stated in the outset, the time complexity of building the query-tree depends on the number of different labels that can be attached to the nodes in the tree. We have seen that the number of labels we may have for L_{sat} with the constraint language $\mathcal{L}^{\wedge,\vee}$ is exponential in the arity of the predicates. The number of labels we may have

for L_{min} or L_{neg} may be doubly exponential in the arity. The following theorem shows that we cannot expect to do much better than that. Specifically, it shows that once we introduce the predicate $\neq$, the lower bound on the problem of detecting strong irrelevance is exponential in the arity. The same is true for encoding L_{min} even without interpreted predicates.

Theorem 3.26: *Given a set of rules $\mathcal{P}$, a query predicate q, and a rule $r \in \mathcal{P}$, deciding $SI(r, q, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_q)$ is hard for exponential time if the rules may contain the predicate $\neq$.*

Deciding $SI(r, q, \Sigma_{\mathcal{P}}, DI_2, M1)$, is hard for exponential time even if $\mathcal{P}$ does not contain any interpreted predicates.

The proof is based on reducing the acceptance problem of a linear-space alternating Turing machine (ATM) to the problem of detecting strong irrelevance of rules. The details of the proof are given in Appendix A.

3.6 Summary

In this chapter we presented a general method for encoding a set of derivations, therefore enabling us to deduce properties of that set efficiently. Specifically, the method enables us to deduce strong irrelevance claims. The method involves constructing a query-tree that finitely encodes all the possible derivations in the given set. The key issue in the construction of the query-tree is its termination condition which is based on a labeling scheme we have devised. The labeling scheme depends on the specific set of derivations we wish to encode. We have shown three instances of the query-tree method: (1) encoding the set of all derivations for Horn rule KBs with interpreted predicates, (2) encoding the set of all minimal derivations of a query and (3) encoding the set of valid derivations when rules may have negated EDB literals in their antecedents. Importantly, in these instances, the number of possible labels and therefore the size of the query-tree, does not depend on the number of rules in the knowledge base, only on the arity of the predicates. Consequently, the query-tree algorithm is likely to scale up to large knowledge bases. In addition to the three instances described, the method provides a powerful conceptual framework in which devising new labeling schemes becomes much easier.

3.6.1 Related Work

The intuition behind the query-tree algorithm comes from translating the problem into a decision problem for tree-automata.[10] In fact, we have argued that a finite

[10] See [Vardi, 1989] for a discussion of the importance of tree automata in database theory.

labeling scheme essentially guarantees that the set of derivations can be recognized by a reachability test on a finite tree automaton. With respect to tree automata, the contribution of our work is twofold. The first is showing that the problem can be recognized by a tree automaton. This involves coming up with the labeling scheme (i.e., the states of the automaton), showing that indeed it is sufficient to encode precisely the set of symbolic derivations of interest, and showing that examining this set of symbolic derivations is enough to decide irrelevance. The second is developing the query-tree which is a more efficient and natural recognizer of the set of symbolic derivations. The query-tree essentially combines the creation of the tree automaton and the reachability test into one algorithm. Moreover, the query-tree will usually produce only a subset of the states of the automaton needed to recognize the set of derivations, and working with the query-tree is conceptually simpler than working directly with tree automata. As we see in the next chapter, the query-tree will also lend itself to other natural usages.

Several other authors have considered static analysis of rules for different purposes, such as explanation based learning [Etzioni, 1993], partial evaluation of logic programs [Smith and Hickey, 1990; Lloyd and Shepherdson, 1991; Bruynooghe *et al.*, 1991], automated reasoning [Kowalski, 1975; Bruynooghe *et al.*, 1989] and deductive databases [Srivastava and Ramakrishnan, 1992; Ullman, 1989]. Some have also used graph-like representations of the rules, such as problem space graphs [Etzioni, 1993], connection graphs [Kowalski, 1975], compilation graphs [Bruynooghe *et al.*, 1989] and rule/goal graphs [Ullman, 1989]. Others have used rule folding/unfolding in their analysis.

The key issue common to work that utilizes graph-like representations of rules or fold/unfold transformations is when to terminate the creation of the graph (or when to stop unfolding the rules). The query-tree is novel in that it gives a well motivated termination criterion based on manipulation of the interpreted constraints that appear in the rules. Consequently, with the exception of [Srivastava and Ramakrishnan, 1992], only the query-tree can be shown to be complete in more than straightforward cases (i.e., in the presence of recursion and constraints). Recall that completeness guarantees that the query-tree encodes *precisely* the set of desired derivations. [Srivastava and Ramakrishnan, 1992] have a similar result to ours[11] but only for the case of L_{sat} (and not for the case of conjunctive order constraints). Their techniques cannot be extended to the cases covered by our general method.

Another important difference is the size of the query-tree, which depends only on the arity of the predicates. In contrast, in previous tree-like structures (e.g., [Etzioni, 1993]), the termination condition of the tree involves checking whether a node is isomorphic to one of its ancestors. This leads to a tree whose size can be exponential in the number of rules.

[11] Obtained simultaneously with ours.

Connection graphs [Kowalski, 1975] were also developed for the purpose of focusing a theorem prover by precomputing all the possible pairs of resolvable clauses. Clearly, if a certain clause appears in a component of the graph that is not connected to the component of the negation of the query, it can be removed from the KB (i.e., it is strongly irrelevant). However, connection graphs only capture a subset of the possible dependencies between clauses. Specifically, they only show that two clauses connected to a link are unifiable, but say nothing about the relationship between clauses connected via longer paths in the graph. Other work [Sickel, 1976; Chang, 1979] has considered following only certain *walks* on the graph, however, these walks are not guaranteed to encode valid derivations, as are the paths encoded in the query-tree.

Chapter 4

Uses of the Query-Tree

The query-tree, as described in the previous chapter, is a powerful tool for relevance reasoning and speeding up inference. In this chapter we describe uses of the query-tree for these purposes. Section 4.1 describes two uses of the query-tree for speeding up inference. In the first, the query-tree is used to decide which ground formulas are strongly irrelevant to the query. Based on that determination, we create specialized database indices that see only the ground formulas that are (possibly) relevant to a class of queries. Using these indices for fetching ground formulas significantly speeds up inference. The second use of the query-tree is based on the observation that the tree also encodes all the possible sequences of rule applications and database lookups that can result in derivations of the query. We can therefore use the query-tree to guide the search of a backward chainer to follow only these sequences. We present and analyze experimental results which show that both these uses yield significant savings in practice.

Section 4.2 considers the problem of deriving logical conclusions from irrelevance claims that are given to the system by an external source. It describes an algorithm based on the query-tree for deriving such conclusions. It also describes an algorithm that uses the query-tree to derive logical conclusions from *relevance-claims*, i.e., claims that state that certain formulas are *necessarily* used in derivations of the query. Finally, Section 4.3 describes how the query-tree can be used to extend other query evaluation methods.

4.1 Using The Query-Tree to Speed Up Inference

The first use of the query-tree is based on the observation (Corollary 3.7) that it tells us exactly which formulas may be relevant to a query (or set of queries). Specifically, a rule is strongly irrelevant to the query if and only if it does not appear in the tree. A

ground formula is strongly irrelevant if and only if it does not satisfy the constraint-label of any goal-node in the tree with which it can be unified. Consequently, when answering the query, these rules and ground formulas can be ignored. For instance, in the *goodPath* example (repeated in Figure 4.1), the rule r_5 can be ignored. Similarly, formulas of the relation *step* that do not satisfy $\{100 < X < Y < 170\}$ can also be ignored.

We can use this property of the query-tree to speed up inference for sets of queries that occur frequently. Given such a set of queries, we build a query-tree for it and create specialized indices *only* on the formulas that are not strongly irrelevant to queries in the set. The cost of preprocessing the knowledge base in such a way involves the cost of building the query-tree and the cost of one pass over the knowledge base to build the specialized indices. However, the payoff of removing irrelevant formulas can be significant because the size of the space that an inference mechanism needs to search can be drastically reduced. Specifically, it is guaranteed that every time a ground formula is retrieved, the formula may be part of a derivation of the query since it satisfies the constraint label of some node in the query-tree. This is especially significant when lookups are made with some unbound variables. For instance, in our example, there will be many lookups of the form $step(a, Y)$, where a is some constant and Y is unbound. Using the specialized index on the formulas of the predicate *step* guarantees that every formula retrieved will satisfy $\{100 < Y < 170\}$. In contrast, retrieving a formula that does not satisfy this constraint can generate a whole search subtree that is guaranteed to be useless. Note that even if the reasoning mechanism detects immediately (by checking the available constraints) that the retrieved formula is irrelevant, the cost of doing all the useless lookups and checking the constraints can be arbitrarily large.[1]

The second use of the query-tree is based on the observation that the tree also encodes the *sequences* of rule applications and database lookups that can result in derivations of the query. We can use this observation to further control our search. To illustrate, consider the following example.

Example 4.1: Consider a knowledge base defining a relation *dessertMeal* with the following rules. Its query-tree is shown in Figure 4.2.

r_1 : $cheapMeal(D_1, W_1) \wedge meat(D_1) \wedge expensiveMeal(D_2, W_2) \wedge dessert(D_2) \Rightarrow$
$dessertMeal(D_1, W_1, D_2, W_2)$
r_2 : $dish(X, Z) \wedge (Z \leq 15) \wedge compatible(X, Y) \Rightarrow cheapMeal(X, Y)$
r_3 : $dish(X, Z) \wedge (Z > 15) \wedge compatible(X, Y) \Rightarrow expensiveMeal(X, Y)$
r_4 : $beef(X) \wedge redWine(Y) \Rightarrow compatible(X, Y)$
r_5 : $dessert(X) \wedge sweetWine(Y) \Rightarrow compatible(X, Y)$

[1] Note that in order to detect irrelevant formulas immediately, the reasoning mechanism must propagate the constraints in the same fashion done in creating the query-tree.

The knowledge base Δ consists of the following rules:

$r_1 : badPoint(X) \wedge path(X, Y) \wedge goodPoint(Y) \Rightarrow goodPath(X, Y).$
$r_2 : link(X, Y) \Rightarrow path(X, Y).$
$r_3 : link(X, Z) \wedge path(Z, Y) \Rightarrow path(X, Y).$
$r_4 : step(X, Y) \Rightarrow link(X, Y).$
$r_5 : bigStep(X, Y) \Rightarrow link(X, Y).$

The following constraints are given on the ground facts:

$badPoint(X) \Rightarrow 100 < X < 200.$
$step(X, Y) \Rightarrow X < Y.$
$goodPoint(X) \Rightarrow 150 < X < 170.$
$bigStep(X, Y) \Rightarrow X < 100 \wedge Y > 200.$

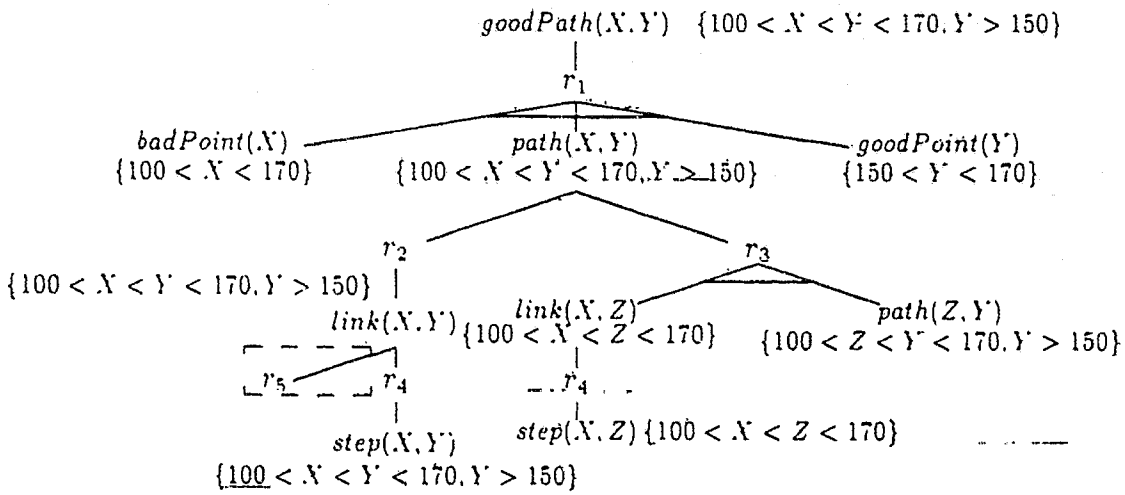

Figure 4.1: The query-tree for *goodPath*.

The predicates *meat*, *beef* and *dessert* are sort predicates (*dessert* is disjoint from the other two). The relation *compatible* represents pairs consisting of a wine and a dish that are compatible with each other. The relation *dish* represents the available dishes and their prices. Consider formulas of the relation *dish*. Any formula that satisfies either $(beef(D_1) \wedge Z \leq 15)$ or $(dessert(D_2) \wedge Z > 15)$ may be relevant to the query *dessertMeal*. However, as a subgoal of r_2, we need only consider formulas of *dish* that satisfy the first constraint, whereas as a subgoal of r_3, only formulas that satisfy the second constraint are needed. Moreover, the query-tree shows that rule r_4 can only be applied to a subgoal of r_2, and not of r_3 (and vice versa for r_5). ∎

To exploit this additional control knowledge, we create specialized indices for every leaf in the query-tree and modify the inference mechanism to follow *only* the paths permitted by the query-tree. In our example, we create one index for beef dishes under $15 and another for dessert dishes over that price. To follow the query-tree

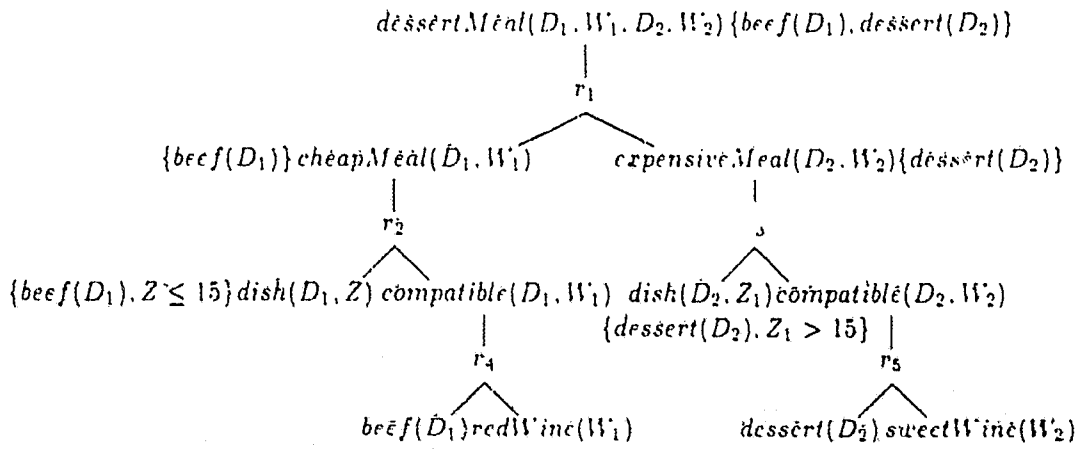

Figure 4.2: Avoiding search paths using the query-tree

during inference, we attach to every subgoal n in our search a node in the query-tree $\phi(n)$. We start by assigning the root of the query-tree to the query. At every step, if n is a database lookup subgoal (i.e., a subgoal of an EDB predicate), we perform the lookup using the specialized index of $\phi(n)$. Otherwise, we expand n only with the rules that are children of the expanded equivalent of $\phi(n)$.[2] We assign to the subgoals of n the appropriate subgoals of the rule-node in the query-tree. As a result, the inference engine follows only the paths encoded by the query-tree and in every database lookup it retrieves only ground formulas that can be used in derivations in the current path.

4.1.1 Experimental Results

The impact of the savings achieved by using the query-tree were tested using a depth first search backward chainer on Horn rules.[3] Given a knowledge base Δ and a query schema q (i.e., a query with free variables), we built a query-tree for q and two sets of indices on ground formulas in Δ. The first set $\mathcal{I}_1$ included an index on every relation that includes only the formulas that were deemed not strongly irrelevant by the query-tree. Specifically, a ground formula $e(a_1, \ldots, a_n)$ is included in the index for the relation e in $\mathcal{I}_1$ if $a_1, \ldots, a_n$ satisfies the constraint label of *some* leaf of the

[2] Which may be the node $\phi(n)$ itself.

[3] The performance of the backward chainer compared favorably with that of Epikit (a commercial implementation of MRS [Russell, 1985]). Furthermore, the speedups attained by removing irrelevant formulas (BC2 below) were also tested using the backward chainer of Epikit and the speedups attained were even better than those reported here. In the experiments we tested several rule and goal orderings. The results are shown for the ordering that yielded the best results consistently for *all* three versions of the backward chainer

predicate c in the query-tree.[4] The second set of indices $\mathcal{I}_2$ included one index for every EDB leaf in the query-tree. We measured three running times:

- **BC1** - the backward chainer on Δ using the original indices in the KB.
- **BC2** - the backward chainer on Δ using the indices $\mathcal{I}_1$, i.e., ignoring strongly irrelevant formulas.
- **BC3** - A backward chainer that uses the indices $\mathcal{I}_2$ and only follows the paths allowed by the query-tree.

We tested over 20 query schemas taken from the following four domains:

1. A travel domain using a database of real airline data describing flights between cities in the U.S (examples 3-6 in the tables).
2. A wine domain consisting of a knowledge base of 50 rules describing various wines and dishes and compatibilities between them (based in part on [Rombauer and Rombauer-Becker, 1975]) (examples 7-8).
3. A student-advisor domain using a knowledge base about computer science Ph.D graduates, including advisor, school and graduation dates (examples 9-10).
4. The *goodPath* example, using the rules in Example 4.1 (examples 1-2).

The first and fourth domains usually yield deep recursive search trees, even though the number of rules is small. The second domain is non-recursive and yields shallow but bushy (i.e., large branching factor) search trees. In the third domain, search trees have a low branching factor (which was from student to advisor).

Table 4.1 presents the results of the experiments for the case where we are looking for all solutions to a query (e.g., find all X, Y such that $goodPath(X, Y)$ is derivable). In the table, *Filtering Time* includes the time taken to build the query-tree and create all the indices (both $\mathcal{I}_1$ and $\mathcal{I}_2$). *Percent irrelevant* is the percent of ground formulas in the knowledge base that were deemed strongly irrelevant (and therefore not included in $\mathcal{I}_1$). The next columns show the time taken to find all the solutions to the query. The respective running times of BC1, BC2 and BC3 are shown, as well as the ratios of running times. In addition to measuring running times, the number of nodes expanded in the search was also counted. The last two columns show the ratios of the number of nodes expanded by BC2 and BC3 compared to BC1.

The results show significant speedups for both **BC2** and **BC3**. For **BC2**, the speedups were usually in excess of a factor of 3, ranging up to 31 (mean: 10.4). The results

[4] Note that the original knowledge base had an index for each ground relation

show that by following the query-tree using BC3 we often get additional improvements. The speedups of BC3 over BC1 were usually in excess of 5, ranging up to 1190 (mean: 41, excluding example #6).[5] In terms of nodes expanded, the average speedup for BC2 was 10, while the average speedup for BC3 was 37 (excluding example #6). The results clearly show that if we are looking for all solutions to the query, building the query-tree and the specialized indices will yield significant savings.

	KB size		Filtering Time	Percent irrelevant	Solution time (sec.)					Nodes expanded	
	Facts	Rules	(sec.)		BC1	BC2	$\frac{BC1}{BC2}$	BC3	$\frac{BC1}{BC3}$	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$
1.	350	6	1.8	63	2780	182	15	183	15	10.5	10.5
2.	350	6	1.8	63	618	231	2.7	233	2.7	2.5	2.5
3.	200	18	6.5	69	372	14	27	8.6	43	22	28
4.	200	18	5.6	69	25	5.5	4.5	4.5	5.6	4.7	6
5.	200	18	20.7	64	3975	205	19	13	306	17	295
6.	200	18	14.7	68	1278	41	31	1.1	1190	31	1630
7.	1300	47	25	59	8740	8720	1	363	24	1	14
8.	1300	47	11.6	60	50	42	1.2	11	4.5	1.2	2.8
9.	150	17	0.8	59	35	7.5	4.6	7.5	4.6	4.5	4.5
10.	150	17	0.6	59	2.8	0.4	7.6	0.4	7.6	4.8	4.8

Table 4.1: Experimental results: finding all solutions.

Table 4.2 presents the results of the experiments for the case in which we use the query-tree built for a query schema to solve ground queries or to find the *first* solution to a query with free variables (i.e., the query-tree was built for $goodPath(X, Y)$ and the query is $goodPath(130, 160)$, or we are trying to find the first binding for X and Y). The second and third columns show the ratios of the number of nodes expanded for ground queries. The next columns show the node ratios of finding the first solution to the query. The next column compares the preprocessing time and the time to find solutions to the query. It shows the number of calls (each looking for the *next* solution) after which the preprocessing time equals the time to answer the queries. The last column shows the number of solutions found for the query. The results indicate that often the preprocessing pays off after a very small number of solutions and therefore it is beneficial to build a query-tree even in cases when we are searching for few solutions.

4.1.2 Analysis

The experiments showed that the savings achieved by using the query-tree are affected by several factors. In this section we describe these effects.

[5] Example #6 was excluded from the mean because the speedups it yielded were exceptionally high

	Ground queries		Find first solution		solutions needed	number of
	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$	to break even	solutions
1.	5.8	6.1	85	85	1	187
2.	1.8	1.8	4.5	4.5	1	187
3.	33	74	2.5	2.6	1	49
4.	4.6	7	6.1	6.2	6	37
5.	15	290	1	1	1	12
6.	33	1550	31	1630	1	0
7.	1	2	1.4	25	115	41000
8.	1.1	1.4	1.6	4.5	81	420
9.	1.3	1.3	123	123	2	353
10.	1	1	4.8	4.8	1	0

Table 4.2: Experimental results: ground queries and finding the first solution.

Percent of Irrelevant Formulas

The analysis of the algorithm suggest that the speedups obtained will be significantly affected by the percent of formulas in the knowledge base that are found to be irrelevant to the query. To test this effect, we ran several variants of each example, that differed only in the constants appearing in the rules (which had the effect of varying the percent of irrelevant formulas). The results, shown in Table 4.3, show that the speedups grow significantly as the percent of irrelevant formulas increases. For example, when 90% of the facts are found to be strongly irrelevant, we get speedups greater than a factor of 100.

It is important to note that we have the flexibility of building a query-tree at different levels of generality and thereby to achieve varying percents of irrelevant formulas. For example, instead of building a query-tree for the query schema $goodPath(X, Y)$, we can build one for $goodPath(120, Y)$. Doing so will result in deeming additional formulas irrelevant (e.g., $step(X, Y)$ for $100 < X < 120$ in this case). However, the indices created by this query-tree will be usable for a smaller set of queries. Consequently, in using the query-tree one should attempt to identify the most accurate characterizations of frequently occurring sets of queries.

Example 1			Example 3			Example 9		
Percent	Solution time		Percent	Solution time		Percent	Solution time	
irrelevant	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$	irrelevant	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$	irrelevant	$\frac{BC1}{BC2}$	$\frac{BC1}{BC3}$
45	5	5	21	1.8	3.1	4	1.1	1.1
63	15	15	45	4.5	7.6	15	1.3	1.3
79	101	101	69	27	43	59	4.6	4.6
92	1250	1240	92	381	417	82	23.5	23.5

Table 4.3: Changing the percent of irrelevant formulas

The Number of Ground Formulas in the Knowledge Base

The second factor that affects the speedups that was suggested by the initial results is the number of ground formulas in the original knowledge base. To test this effect, we ran each of the examples with databases containing a different number of ground formulas. The results, shown in Table 4.4, show that the speedups increased as the size of the databases grew, even if the percent of irrelevant formulas remained roughly the same. The growth can be explained by the fact that the cost of backward chaining is more than linear in the number of formulas. Therefore, the effect of removing some constant percent of formulas will be greater when the overall number of formulas is greater. These results are significant in that they suggest that our methods will scale up to large knowledge bases and be even more effective there (recall that the cost of building the query-tree is independent of the number of ground formulas).

Example 1			Example 3			Example 7		
KB size	Solution time $\frac{BC1}{BC2}$	Solution time $\frac{BC1}{BC3}$	KB size	Solution time $\frac{BC1}{BC2}$	Solution time $\frac{BC1}{BC3}$	KB size	Solution time $\frac{BC1}{BC2}$	Solution time $\frac{BC1}{BC3}$
250	12.6	12.4	100	23	31	540	1.3	11.3
350	15	15	200	27	43	930	1	20
550	20	20	300	35	58	1300	1	24

Table 4.4: Changing the size of the database.

Placement of Interpreted Literals in the Rules

A final factor in the speedups achieved from using the query-tree is the way the interpreted literals are placed in the rules. To illustrate, consider the following set of rules defining the existence of a flight (perhaps with stops) between two cities in the country subject to time constraints (given by the constants s_0 and e_0):

$u_1 : p(X, Y, S_1, E_1) \wedge (s_0 \leq S_1) \wedge (e_0 \geq E_1) \Rightarrow timelyConnect(X, Y)$
$u_2 : fl(X, Y, S, E) \Rightarrow p(X, Y, S, E)$
$u_3 : fl(X, Z, S, T) \wedge (T \leq T_1) \wedge p(Z, Y, T_1, E) \Rightarrow p(X, Y, S, E)$

To describe such paths, the rules can also be written as follows:

$v_1 : p(X, Y, S_1, E_1) \Rightarrow timelyConnect(X, Y)$
$v_2 : fl(X, Y, S, E) \wedge (S \geq s_0) \wedge (E \leq e_0) \Rightarrow p(X, Y, S, E)$
$v_3 : fl(X, Z, S, T) \wedge (T \leq T_1) \wedge (S \geq s_0) \wedge (T \leq e_0) \wedge p(Z, Y, T_1, E) \Rightarrow p(X, Y, S, E)$

The difference between the two sets of rules is that the second set is crafted to exploit the constraints entailed by the interpreted constraints on *timelyConnect*. Specifically, whenever we retrieve a flight formula that violates the constraints (i.e., ends later than e_0 or begins before s_0), we will immediately backtrack. In contrast,

when using the first set of rules, we will compute all possible paths (in a bottom up computation) and check the constraints in the last step of the derivation. Consequently, when using the first set of rules, a strongly irrelevant formula may be the root of an arbitrarily large tree, whereas when using the second set, no such tree will be generated. Consequently, removing strongly irrelevant formulas will have a greater effect for a set of rules like the first one. The experimental results confirm this observation. The example pairs 1 & 2 and 3 & 4 are instances of rules differing exactly in this fashion.

Several points should be noted with respect to this issue:

- Although the speedups are significantly bigger using the first set of rules in each pair, we still achieve significant savings even when the rules are carefully crafted such that the constraints are used to control the search.

- Writing rules with such built-in control has many disadvantages ([Clancey, 1983]). It is extremely difficult to write such rules in practice and is a very error-prone task. Consequently, we expect rules would usually be written without such crafting.

- Crafting a set of rules with such built-in control can however be done easily using the query-tree (as we did in Section 3.4). Specifically, we can create a new rule for every rule-node in the query-tree that includes the constraints of that node. The resulting set of rules will be equivalent to the original set with respect to the query predicate (i.e., will produce the same answer regardless of the database of ground facts). However, using the new set, the tightest constraints will be enforced on the bindings immediately when they appear.

Applicability to Other Inference Mechanisms

The experiments described above were done with a depth first search backward chaining inference mechanism. However, the techniques we described can be applied to a wide range of reasoning mechanisms. The first use of the tree, the removal of strongly irrelevant formulas, is independent of the reasoning scheme used. Following the query-tree can also be integrated easily to any reasoning mechanism. The only requirement is that nodes in the search space be associated with nodes in the query-tree, and the particular order in which the space is searched is unimportant.

Finally, there are several possible schemes for integrating the construction of the query-tree and the indices with search for the solution. One possibility is to create a specialized index for a relation only if it is actually referenced in the search. This way one can avoid creating indices that will not be used.

4.2 Irrelevance Claims from an External Source

Until now we have used the query-tree to decide automatically which formulas are irrelevant to a given query. Often a user may be able to supply the system with additional irrelevance claims based on his/her knowledge about the domain and about the ground formulas in the KB (or those that may appear in the KB). Specifically, the user may know that a set of formulas Φ is strongly irrelevant to the query q, given the possible ground formulas that may occur in the knowledge base. This knowledge may not be expressible as explicit constraints on ground formulas, which can be used directly by the query-tree. For example, this knowledge may be based on the fact that the join of two relations is empty, which is not expressible using Horn rules. Alternatively, this knowledge may be heuristic in nature.

Clearly, if we are told that a formula ϕ is strongly irrelevant to q, we can ignore ϕ when answering q. However, we may also be able to conclude that other formulas are irrelevant as well. This section describes an algorithm for deriving such conclusions using the query-tree. In Section 4.2.1, we consider a different kind of external knowledge in which the system is told that some formulas are *necessarily* relevant to the query.

Formally, the problem we consider here is as follows. Suppose that $\mathcal{P}$ is a set of rules and let I be an irrelevance claim stating that a set of formulas Φ is strongly irrelevant to a query q. More precisely, I actually states that the set of possible KBs is *some* subset $\Sigma' \subseteq \Sigma_{\mathcal{P}}$, such that $SI(\Phi, q, \Sigma', DI_2, \mathcal{D}_q)$ holds.[6] We assume that Φ is composed of a set of rules $\Phi_r \subseteq \mathcal{P}$ and a set of ground formulas Φ_g specified as a set $\{p(\bar{X}) \mid C(\bar{X})\}$, where p is some IDB predicate[7] and $C(\bar{X})$ is a formula with only interpreted predicates.[8] Our goal is to find which strong irrelevance claims follow from I, i.e., for which formulas ϕ_1 the following holds:

$$SI(\Phi, q, \Sigma', DI_2, \mathcal{D}_q) \Rightarrow SI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q).$$

To derive logical conclusions from I using the query-tree, our strategy is to create a set of rules $\mathcal{P}_1$, such that when formulas from Φ are excluded, $\mathcal{P}_1$ and $\mathcal{P}$ produce the same derivations of the query q for every set of ground facts G. We then create a query-tree for $\mathcal{P}_1$ and find all formulas that are strongly irrelevant to q. If a formula ϕ_1 is found to be strongly irrelevant with respect to $\mathcal{P}_1$, then it is also strongly irrelevant with respect to $\mathcal{P}$ whenever I holds.

[6] Note that there may be many such subsets Σ'. In our algorithm and analysis we will assume that Σ' the maximal such subset, but our conclusions will hold for any such subset.

[7] Note that the case of EDB predicates can be handled in a straightforward fashion by the query-tree algorithm.

[8] Φ can include a collection of such sets. However, for simplicity of exposition we assume that there is only one

Formally, given that $\Phi = \Phi_r \cup \{p(\bar{X}) \mid C(\bar{X})\}$, the set of rules $\mathcal{P}_1$ is defined as follows:

1. If $r \in \mathcal{P}$ and $r \notin \Phi_r$ and the head of r is not p, then $r \in \mathcal{P}_1$.
2. If $r \in \mathcal{P}$ and r is of the form $q_1(\bar{X}_1) \wedge \ldots q_l(\bar{X}_l) \Rightarrow p(\bar{X})$, $r \notin \Phi_r$ and

$$\neg C(\bar{X}) \equiv D_1(\bar{X}) \vee \ldots \vee d_m(\bar{X}),$$

where each d_i is a conjunction of literals of interpreted predicates; then $\mathcal{P}_1$ includes rules of the form

$$q_1(\bar{X}_1) \wedge \ldots q_l(\bar{X}_l) \wedge d_i(\bar{X}) \Rightarrow p(\bar{X})$$

for $(1 \leq i \leq m)$. For future reference we denote these rules by $\mathcal{P}_1(r)$.

Example 4.2: Suppose that we are told that the set $\Phi = \{path(X, Y) \mid X < 120\}$ is strongly irrelevant to the query *goodPath* in Figure 4.1. The rules in $\mathcal{P}_1$ would include rules r_1, r_4 and r_5, as well as the following rules for the predicate *path*:

$link(X, Y) \wedge (X \geq 120) \Rightarrow path(X, Y)$.
$link(X, Z) \wedge (X \geq 120) \wedge path(Z, Y) \Rightarrow path(X, Y)$.

The query-tree for $\mathcal{P}_1$ will show that the formulas $\{badPoint(X) \mid X < 120\}$ are strongly irrelevant to $goodPath(X, Y)$. ∎

To prove the correctness of our algorithm, we show that $\mathcal{P}_1$ produces precisely the same derivations as those produced by $\mathcal{P}$, except for derivations including formulas in Φ:

Lemma 4.3: *Let $\mathcal{P}$ be a set of Horn rules and let $\mathcal{P}_1$ be the set of rules produced by our algorithm, given that the formulas Φ are irrelevant to the query. Let D be an arbitrary set of ground facts.*

A derivation d that does not use formulas from Φ is a valid derivation of the query from $D \cup \mathcal{P}$ if and only if there is a valid derivation d' of the query from $D \cup \mathcal{P}_1$, such that the only difference between d and d' is that every instance of a rule r of p used in d is replaced in d' by an instance of a rule in $\mathcal{P}_1(r)$.

Proof: Let d' be a derivation of the query from $D \cup \mathcal{P}_1$. Clearly, if we replaced each of the rules of p used in d' by its original rule in $\mathcal{P}$, the resulting derivation would be a valid derivation of the query from $\mathcal{P}$ because the original rule does not contain the additional literal of the interpreted predicate (d_i) in the antecedent.

Conversely, let d be a derivation of the query from $D \cup \mathcal{P}$. Since d does not use formulas from Φ, it does not include rules from Φ_r. To complete the proof, suppose

the formula $p(\bar{a})$ is used as a part of the derivation d and was derived using the instantiated rule $q_1(\bar{a}_1) \wedge \ldots q_l(\bar{a}_l) \Rightarrow p(\bar{a})$. Since $p(\bar{a}) \notin \Phi_g$, there must be some i, such that $\bar{a}$ satisfies d_i. Therefore, $p(\bar{a})$ can also be derived using the instantiated rule $q_1(\bar{a}_1) \wedge \ldots q_l(\bar{a}_l) \wedge d_i(\bar{a}) \Rightarrow p(\bar{a})$, which is an instance of a rule in $\mathcal{P}_1$.

Consequently, every rule of p in d can be replaced by a rule of $\mathcal{P}_1$, and so we can construct a derivation for the query from the rules of $\mathcal{P}_1$. ∎

The following corollary shows that inferring strong irrelevance claims from the query-tree of $\mathcal{P}_1$ is sound:

Corollary 4.4: *Let $\mathcal{P}_1$ be the set of rules constructed from $\mathcal{P}$ and the irrelevance claim I stating that Φ is strongly irrelevant, i.e., $I = SI(\Phi, q, \Sigma', DI_2, \mathcal{D}_q)$. If ϕ_1 is a ground atomic formula and $SI(\phi_1, q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, then*

$$I \Rightarrow SI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$$

holds. If ϕ_1 is a rule, and for all $r' \in \mathcal{P}_1(\phi)$, $SI(r', q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, then

$$I \Rightarrow SI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$$

holds.

Proof: By Lemma 4.3, a formula ϕ_1 can be used in a derivation of the query q from $\mathcal{P}$ and some database D if and only if either:

1. ϕ_1 is used in a derivation of q from $\mathcal{P} \cup D$ that *includes* some formula from Φ or
2. ϕ_1 is a ground atomic formula and can be used in a derivation of q from $\mathcal{P}_1 \cup D$ or
3. ϕ_1 is a rule and there is some $r' \in \mathcal{P}_1(\phi_1)$ that can be used in a derivation of q from $\mathcal{P}_1 \cup D$.

Consequently, if Σ' is a set of databases in which the formulas Φ are strongly irrelevant to q, then the first possibility is ruled out. In the case of ϕ_1 being a ground atomic formula, since $SI(\phi_1, q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, the second possibility is ruled out, and if ϕ_1 is a rule, then because all the rules in $\mathcal{P}_1(\phi_1)$ are not in the query-tree, the third possibility is ruled out. Consequently, the corollary holds. ∎

The same algorithm can be used to derive logical conclusions from external weak irrelevance claims.

Corollary 4.5: *Let $\mathcal{P}_1$ be the set of rules constructed from $\mathcal{P}$ and the set of irrelevant formulas Φ. Suppose $I_w = WI(\Phi, q, \Sigma', DI_2, \mathcal{D}_q)$. If ϕ_1 is a ground atomic formula and $SI(\phi_1, q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, then*

$$I \Rightarrow WI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$$

holds. If ϕ_1 is a rule, and for all $r' \in \mathcal{P}_1(\phi)$, $SI(r', q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, then

$$I \Rightarrow WI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$$

holds.

Proof: Suppose I_w holds. This implies that for any database $D \in \Sigma'$, any answer to the query has a derivation d that does not use formulas in Φ. Therefore, by Lemma 4.3, there will be a derivation d' of the query from $\mathcal{P}_1 \cup D$ corresponding to d. If ϕ_1 is a ground formula, then since $SI(\phi_1, q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds, then d' does not use ϕ_1, and therefore d does not use ϕ_1 (because d' contains a superset of the ground formulas in d). Similarly, if ϕ_1 is a rule, then since $SI(r', q, \Sigma_{\mathcal{P}_1}, DI_2, \mathcal{D}_q)$ holds for every rule r' in $\mathcal{P}_1(\phi_1)$, d does not use ϕ_1. ∎

Our inference procedure is not complete. In fact, in general, it is not possible to find *all* the consequences of an irrelevance claim I, even if Φ includes a single rule.

Theorem 4.6: *Suppose that $\mathcal{P}$ is a set of function-free Horn rules with no interpreted predicates. Let I be the irrelevance claim stating that a rule $r \in \mathcal{P}$ is strongly irrelevant to the query q, i.e., the set of possible KBs is Σ', where Σ' is the maximal subset of $\Sigma_{\mathcal{P}}$ such that $SI(r, q, \Sigma', DI_2, \mathcal{D}_q)$ holds. There is no algorithm that will determine whether $SI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$ holds for an arbitrary formula ϕ_1.*

The proof is based on a reduction from the rule redundancy problem, and its details are given in Appendix A.

Example 4.7: As an example of a knowledge base for which our algorithm will not find all the conclusions of I, consider the following rules:

$s_1 : e(X) \Rightarrow p_1(X)$
$s_2 : e(X) \Rightarrow p_2(X)$
$s_3 : p_1(X) \Rightarrow p(X)$
$s_4 : p_2(X) \Rightarrow p(X)$

Suppose I states that rule s_1 is strongly irrelevant to p. Our algorithm will build a query-tree that will consist of the rules s_2 and s_4 and will deem s_3 to be strongly irrelevant. However, since the derivations using s_1 are isomorphic to the derivations using s_2, strong irrelevance of s_1 implies strong irrelevance of s_2 and s_4 as well. ∎

As can be seen in this example, we can combine our algorithm with algorithms for deriving weak irrelevance claims (see Chapter 5), and the resulting algorithm will be able to derive additional irrelevance claims. Furthermore, using the results described in Sections 3.3 and 3.4, we can also use the query-tree to derive additional strong irrelevance claims by looking only at minimal derivations and in knowledge bases that include some forms of negation in their antecedents.

4.2.1 Relevance Claims

A different kind of knowledge that a user may be able to provide a system is positive *relevance* knowledge. For example, the user may know that a certain formula *must* be used in every possible derivation of the query. Such knowledge may be available in several contexts.. For example, (as we often see in textbooks), we may be given a hint that a certain lemma must be used in a proof of a theorem. As another example, suppose a new formula is added to a knowledge base, and we want to find the new derivable conclusions. In such a case, we know that the derivations of the new conclusions must include the updated formula. As in the case of external irrelevance claims, we may be able to use a relevance claim in order to deduce that some other formulas are irrelevant to the query. In this section we show how to use the query-tree to deduce such conclusions. In theory, it is possible to construct a space of definitions for relevance analogous to the space we constructed for irrelevance. However, here we consider only one such definition:

Definition 4.8: A formula ϕ is relevant to a query q with respect to a set of knowledge bases Σ, denoted $Relevant(\phi, q, \Sigma)$, if ϕ appears in every derivation of q from each of the KBs in Σ. ∎

To derive irrelevance-claims that are logical consequences of a given relevance-claim we rely on the following observation. If a formula ϕ_1 cannot appear in any derivation that includes ϕ, and ϕ is known to be relevant to the query q, then ϕ_1 must be strongly irrelevant to q. The query-tree enables us to find such relations between formulas in the knowledge base, formalized by the following *exclusiveness* condition:

Definition 4.9: Two formulas ϕ_1 and ϕ_2 are said to be *exclusive* with respect to a set of rules $\mathcal{P}$ if there is no set of ground formulas G such that there is a derivation of an answer to the query from $\mathcal{P} \cup G$ that uses both ϕ_1 and ϕ_2. ∎

ϕ_1 and ϕ_2 being exclusive is a sufficient condition for deriving strong irrelevance. The following lemma follows from the definitions:

Lemma 4.10: *If Σ' is a set of databases such that Relevant(o_1, q, Σ') holds, and o_1 and o_2 are exclusive, then $SI(o_2, q, \Sigma', DI_2, \mathcal{D}_q)$ holds.*

Proof: If o_1 appears in every derivation of q from databases in Σ', and o_1 and o_2 cannot appear in the same derivation, then o_2 does not appear in any derivation of q from databases in Σ'. ∎

The exclusiveness condition can be determined using the query-tree. Figure 4.3 describes an algorithm that finds all the formulas Φ such that Φ and r are exclusive with respect to $\mathcal{P}$, where r is a rule.[9] Informally, the algorithm begins from every appearance, r_0, of r in the query-tree and marks all the nodes that can appear in a derivation together with r_0. It labels *above* any node that can appear above r_0 in a derivation tree, and labels *below* any node that can appear in such a tree, but not necessarily above r_0. The correctness of the algorithm is established by the following theorem.

procedure find-exclusive-formulas(T_0, r)
begin /* T_0 is the query-tree for the rules $\mathcal{P}$. */
  for every appearance, $r_0 \in T_0$ of r **do:**
    repeat
      label r_0 *above* and *below*.
      1: **if** a rule-node n is marked *above*, label its father goal-node *above*.
      2: **if** a goal-node n is marked *above*,
          then label its father *above* and its siblings *below*.
      3: **if** a goal-node n is labeled *above*, label *above* any of its unexpanded equivalents.
      4: **if** n is marked *below* label its children *below*.
      5: **if** a goal-node n is marked *below* **and** m is its expanded equivalent, label m as *below*.
    until no new nodes are marked.
    Any node that has an instance marked *above* or *below* is marked *non-exclusive*.
    Remove all *above* and *below* markings.
  end for
end.

Figure 4.3: Algorithm for finding exclusive rules.

Theorem 4.11: *Given a set of rules $\mathcal{P}$, a query q and a rule $r \in \mathcal{P}$, procedure* **find-exclusive-formulas** *will mark an instance of rule s in the query-tree if and only if s is not exclusive with respect to r. The rule r and the atom $p(a_1, \ldots, a_n)$ are exclusive if $a_1, \ldots, a_n$ does not satisfy the constraint label of any node of p that was marked non-exclusive.*

[9] The algorithm can be extended in a straightforward manner to the case where r is a set of formulas.

Proof: We prove the "only if" part by showing that if a node n was marked *non-exclusive* by the algorithm when the marking began with an appearance r_0 of r, then there is a symbolic derivation tree d encoded by the query-tree (by a mapping ψ) such that there are two nodes n_1 and n_2 in d such that $\psi(n_1) = n$ and $\psi(n_2) = r$.

We prove the claim by induction on the order of the marked goal-nodes. Specifically, we show that for every node n that is marked there is a partial symbolic derivation tree d',[10] encoded by the query-tree using a mapping ψ such that:

- If a node n is marked *above*, then n is $\psi(root(d'))$ and r appears in d'.
- If a node n is marked *below*, then there is a leaf in d', n', such that $\psi(n') = n$, and d' includes r.

As a consequence of these claims, we can show that an appropriate symbolic derivation tree d exists. Given the partial tree d', we consider a symbolic derivation tree encoded by the query-tree in which one of the nodes n is mapped to $\psi(root(d'))$. We replace the subtree of n with d'. And complete the leaves of the resulting symbolic derivation arbitrarily (note that Part 2 of Theorem 3.6 guarantees that the completion can be done). The resulting symbolic derivation is encoded by the query-tree and satisfies the requirements.

The claim holds trivially for the base case that includes the node r_0 and its father and children, since (by Part 3 of Theorem 3.6) there is a symbolic derivation tree encoded by the query-tree that includes r_0. The derivation tree d' includes the rule-node in r, its father and children. In the inductive case, there are several cases in which a node m could have been marked, corresponding to the conditionals in the algorithm:

1. The node m is a father rule-node of n (case 2). By the inductive assumption, the node n is a root of d' that includes r. Consider a partial symbolic derivation tree d'_1 created by adding the rule in m as the father of the root of d'. The mapping of the nodes in d' stays unchanged. The root of d_1 is mapped to the father of m (covering case 1 of the algorithm) and the top rule-node is mapped to m. The siblings of n are leaves in d_1 and they are mapped to the siblings of $\psi(root(d_1))$.

2. The node m is an expanded equivalent of a goal-node n (case 3). Since there exists a partial symbolic derivation d' for which $\psi(root(d')) = n$, we can just as well make $\psi(root(d')) = m$, and the encoding conditions of Definition 3.1 still hold.

[10] The derivation d' is partial because its leaves are not necessarily EDB nodes and its root is mapped to an arbitrary goal-node in the query-tree, not necessarily a root.

3. In case 4 of the algorithm, if n is marked *below*, then by the inductive assumption, then there is a leaf n' in d' such that $\psi(n') = n$. Therefore, if r' is a child rule-node of n, we can consider a partial symbolic derivation d_1 in which n' is expanded with the rule in r'. The mapping ψ will map the child of n' to r' and the children rule-nodes to the corresponding children of r'.

4. In case 5 of the algorithm, the same argument of case 4 holds. Consider the derivation d' in which n' is expanded with one of the children of its expanded equivalent, m. Modify ψ so that $\psi(n') = m$.

To complete the proof, we must show that if a node n can appear with the rule r_0 in a symbolic derivation, then it will be marked by the algorithm. Let d be a symbolic derivation encoded by the query-tree and ψ be the mapping of the nodes of d to the nodes in the query-tree. Assume that there is some node $r' \in d$ such that $\psi(r') = r_0$. We need to show that for every $n \in d$, $\psi(n)$ will be marked by the algorithm. Since r_0 is marked *below*, all the nodes $n \in d$ that are below r' will be marked *below* by a combination of conditionals 4 and 5. Let m be the father goal-node of r'. The node $\psi(m)$ will be marked either by 1 or by 3. The father node of $\psi(m)$ will be marked *above* by 2, and $\psi(m)$'s siblings will be marked *below* by 2. Consequently, if r'_1 is the father of m, then for any node n in its subtree, $\psi(n)$ will be marked by the algorithm. We can continue in the same fashion for r'_1, and show that $\psi(n)$ will be marked for every node n that is a descendent of the top rule-node in d. Finally, $\psi(root(d))$ will be marked by 1. ∎

Example 4.12: Consider the following rules:

$r_1 : wants(X, Y, C) \wedge canAfford(X, Y, C) \Rightarrow buys(X, Y, C)$
$r_2 : wants(X, Y, C) \wedge canGetLoan(X, C_1) \wedge (C_1 \geq C) \Rightarrow buys(X, Y, C)$
$r_3 : Sees(X, Y, C) \wedge likes(X, Y) \Rightarrow wants(X, Y, C)$
$r_4 : priceOf(Y, C) \wedge hasCash(X, C) \wedge (C < 100) \Rightarrow canAfford(X, Y, C)$
$r_5 : customer(X, B) \wedge creditLimit(X, B, C) \Rightarrow canGetLoan(X, C)$

The atom $buys(X, Y, C)$ denotes that X will buy item Y at price C. The person X will buy Y only if she wants it. If she does, she will buy it if she has enough cash at hand or if she can get a loan from a bank to cover the expense. The query-tree constructed for $buys(X, Y, C)$ appears in Figure 4.4. In this example, rules r_4 and r_5 are exclusive. Therefore, suppose we are given that r_4 must be used to answer the query $buys(Sue, Y, Z)$ (because we know that Sue used cash for all her purchases lately). The algorithm will mark the nodes in the query-tree as shown in Figure 4.4, and therefore the rules r_5 and r_2 will be deemed strongly irrelevant to the query. ∎

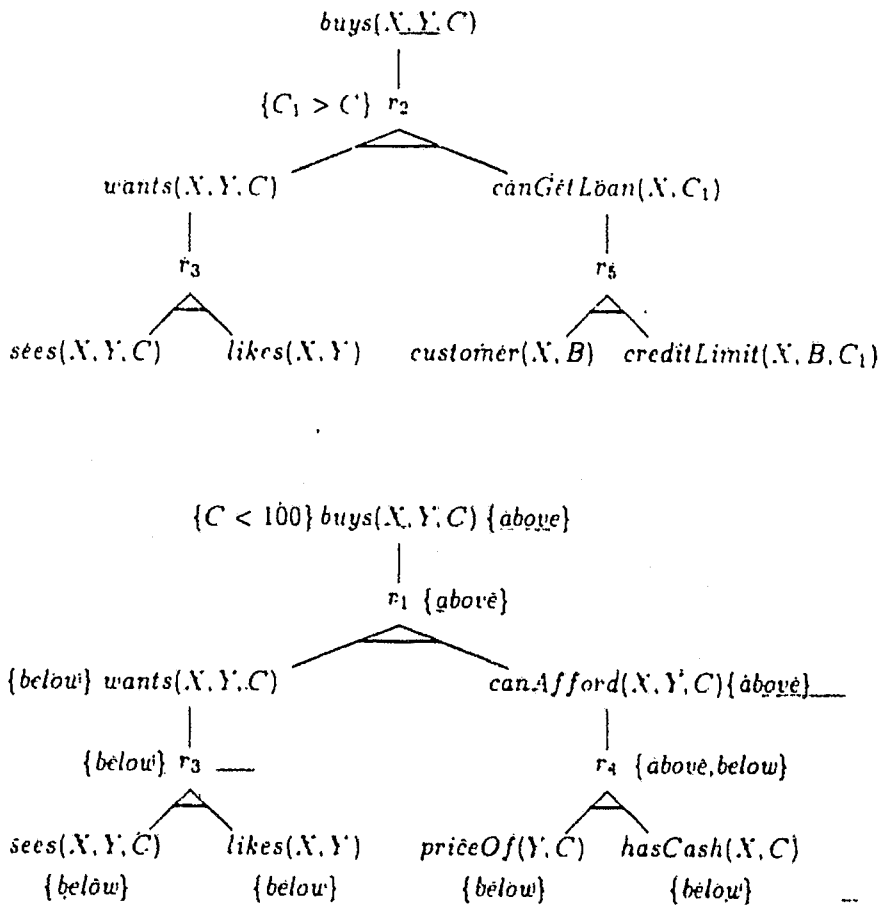

Figure 4.4: The query-tree of Example 4.12.

4.3 Additional Uses of the Query-Tree

In this section we briefly outline several ways in which the query-tree can be used to extend other query evaluation methods.

Combining with Magic Set Transformation

Two primary strategies for evaluating a query with a given set of rules are top-down (e.g., backward chaining) and bottom-up (e.g., forward chaining). Top-down techniques have the advantage that they are more goal-directed, since they exploit the information in the query (e.g., the query-predicate and bindings that appear in the query). However, top-down techniques have the disadvantage that they may result in infinite loops and that they require the operation of unification, instead of the cheaper operation of term-matching, used in bottom-up evaluation. In contrast,

bottom-up techniques will not get into infinite loops (when the rules do not have function symbols), but may compute many facts that are not relevant to the query. The goal of the magic-set transformation method [Ullman, 1989] is to combine the advantages of top-down and bottom-up evaluation methods. It transforms a given set of rules $\mathcal{P}$ to a new set $\mathcal{P}_1$, such that $\mathcal{P}_1$ is equivalent to $\mathcal{P}$ with respect to the query predicate. Moreover, a bottom up evaluation of $\mathcal{P}_1$ provides the goal-directedness focus achieved by top-down evaluation. To illustrate, consider the following rules for transitive closure with additional interpreted constraints:

$r_1 : c(X, Y) \wedge (X < Y) \Rightarrow p(X, Y)$
$r_2 : c(X, Z) \wedge (X < Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$

and suppose our query is to find all Y such that $p(a, Y)$ is derivable, where a is some constant. The transformed set of rules will be the following:

$s_1 : m_p(a).$
$s_2 : m_p(X) \wedge c(X, Z) \wedge (X < Z) \Rightarrow m_p(Z)$
$s_3 : m_p(X) \wedge c(X, Z) \wedge (X < Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$
$s_4 : m_p(X) \wedge c(X, Y) \wedge (X < Y) \Rightarrow p(X, Y)$

The predicate-m_p is the "magic predicate of p" and is used to constrain the tuples that will be computed in the bottom-up evaluation of the rules. Essentially, m_p is the set of constants that *may* appear in the first argument of facts of p that are used in a derivation of $p(a, Y)$.

The limitation of the magic-set transformation is that it can only use binding information in the query. In the above example, it was able to use the fact that the first argument of p in the query is bound to a. However, it cannot use information about constraints on possible bindings. For example, if the query was $p(a, Y) \wedge (Y < 2)$, the magic set transformation would not be able to exploit the constraint $Y < 2$.

The query-tree effectively pushes such constraints from the query to the other rules of the knowledge base. Consequently, we can use it in order to extend the magic set transformation with constraint pushing. Specifically, we can attach *bf* adornments to labels of the nodes in the query-tree. The adornment specifies which arguments of the node are free and which are bound. Adornments can be determined in a top-down fashion as in the magic-set transformation algorithm. We can then refine the equivalence relation on nodes in the tree by requiring that the adornments be the same. The resulting rules will have the *unique-binding* property needed in order to create a magic program (see [Ullman, 1989], Algorithm 13.1, pg. 828). Applying the magic-sets rule transformation to this set of rules will yield the following rules for the query $p(a, Y) \wedge (Y < 2)$:

$s'_1 : m_p(a).$
$s'_2 : m_p(X) \wedge e(X, Z) \wedge (X < Z < 2) \Rightarrow m_p(Z)$
$s'_3 : m_p(X) \wedge e(X, Z) \wedge (X < Z < 2) \wedge p(Z, Y) \Rightarrow p(X, Y)$
$s'_4 : m_p(X) \wedge e(X, Y) \wedge (X < Y < 2) \Rightarrow p(X, Y)$

Note that with these rules, facts of the form $p(X, Y)$ with $Y \geq 2$ will not be produced in a bottom-up computation. Since the rules created by the query-tree are equivalent to the original rules with respect to the query, it follows from Theorem 13.1 in [Ullman, 1989] that our transformation is correct.

It should be noted that using the query-tree to propagate the constraints has an advantage over previous techniques, such as the use of *bcf* adornments [Mumick *et al.*, 1990]. That technique attaches a *c* adornment to an argument of a goal-node if there is *some* known constraint on it. In contrast, the query-tree considers the semantics of the interpreted literals to compute the actual constraint on the arguments.

Message-Passing Query Evaluation Schemes

In a *message passing* scheme for query evaluation [Van-Gelder, 1986], query evaluation is viewed as a system of cooperating processes communicating by message passing. Each process computes some set of tuples (essentially a subset of the relation for some relation). The messages between the processes represent the *needs* of a process and the solutions it generates. A *need* message is generated by a process that needs some relation in order to compute its output relation. For example, a process computing the relation $e_1(1, Y) \bowtie e_2(Y, Z)$ will send a message to a process computing the relation e_1, specifying that it needs the subset of e_1 with the first argument bound to 1. After computing the desired relation, the latter process will send a solution message, with the possible bindings of Y.

The main advantage of a message passing scheme is that by breaking up the problem to such modules with well defined interfaces, we are able to exploit existing operating system features in order to facilitate and speed query evaluation. Such features include scheduling, message passing, and multi-tasking. This scheme is also a first step towards parallel implementation of query evaluation.

To facilitate such an evaluation scheme, we need to determine what processes will exist and how they will communicate. To do so, Van Gelder uses a rule-goal tree which resembles a simple version of the query-tree. Each goal-node in the tree is considered to be a different process in the system. The termination condition that he uses in constructing the rule-goal tree depends only on an isomorphism of the variable patterns of the goal-nodes and of the adornments. Consequently, information about interpreted literals is not propagated and therefore not used to determine the set of processes.

The query-tree can be used directly to extend Van Gelder's scheme to incorporate knowledge about interpreted literals. We can simply refine the labels of the nodes in the query-tree with the adornments used in his construction. As a result, we will be able to distinguish between parts of a relation that can be computed in parallel and are independent of each other. For instance, consider the rules of Example 4.1 repeated below.

$r_1 : cheapMeal(D_1, W_1) \wedge meat(D_1) \wedge expensiveMeal(D_2, W_2) \wedge dessert(D_2) \Rightarrow dessertMeal(D_1, W_1, D_2, W_2)$
$r_2 : dish(X, Z) \wedge (Z \leq 15) \wedge compatible(X, Y) \Rightarrow cheapMeal(X, Y)$
$r_3 : dish(X, Z) \wedge (Z > 15) \wedge compatible(X, Y) \Rightarrow expensiveMeal(X, Y)$
$r_4 : beef(X) \wedge redWine(Y) \Rightarrow compatible(X, Y)$
$r_5 : dessert(X) \wedge sweetWine(Y) \Rightarrow compatible(X, Y)$

Ordinarily, we would have one process for the predicate *dish* that would send its answers to processes of *cheapMeal* and *expensiveMeal*. However, constructing a message passing scheme based on the query-tree will result in two processes for *dish*, one computing cheap beef dishes (and sending its answers to the process *cheapMeal*) and the other computing expensive dessert dishes. As a result, the cost to compute the joins (in rules r_2 and r_3) is significantly reduced.

Deriving Optimal Search Strategies

The query-tree implicitly encodes the space of derivations that an inference engine should search. The novelty of the query-tree is that it encodes a subset of the space that would have been searched by ordinary backward chaining, and therefore following the query-tree enables pruning of parts of the search space. A different approach that was considered to speeding up inference is finding optimal strategies for searching a given space [Smith, 1986; Greiner, 1991; Greiner, 1992].

The query-tree can be used to complement and extend these methods in two ways. First, by delimiting the actual space that needs to be searched, some search paths can be eliminated from consideration when looking for the optimal search strategy. Second, the methods described by Smith and Greiner require a graph-like representation of the possible derivations of the query. The query-tree provides such a representation which treats recursion and interpreted literals in a principled way, unlike the representations that are currently used. Consequently, it can be used as a basis for extending such techniques to fully incorporate knowledge about interpreted literals. In particular, the query-tree can be used to extend Greiner's algorithm [Greiner, 1991] for knowledge bases with recursive rules.

The goal of Explanation Based Learning [Minton *et al.*, 1989] is also to speed up inferences. In EBL, new rules are added to the knowledge base that compress sequences of inference into a single rule. The sequences are learned by examining

derivations of observed queries. The key issue in this approach is the utility of the added rules [Minton, 1988; Etzioni and Minton, 1992]. Adding too many rules may have the inverse effect of slowing down inference. Moreover, the learned rules may be long and require many unification operations. Etzioni [Etzioni, 1993] has shown that much of the speedups obtained by EBL can be obtained by merely doing static analysis of the rules in the knowledge base. Using a tree-like representation of the rules in the knowledge base, called the *Problem Space Graph* (PSG), he showed how to glean from it new rules that were more effective than those learned by standard EBL techniques.

The problem space graph is similar in principle to the query-tree. However it does not consider the semantics of interpreted literals in the rules. It also uses a very simple termination condition in the case of recursion; a node is not expanded if it is a variable renaming of one of its ancestors. It is therefore possible to extend Etzioni's techniques by refining the construction of the PSG with the labeling schemes employed by the query-tree. By attaching constraint labels to the nodes, we can discover additional sequences of actions that are guaranteed to fail. We can also attach tag-labels to nodes and use them to find sequences of actions that necessarily contain loops. A key difference between the PSG and the query-tree is that the decision whether to expand a node in the PSG depends partially on its ancestors. In contrast, the information used in order to decide whether to expand a node in the query-tree is encoded in the node itself. To fully integrate the query-tree and the PSG we need to find methods to terminate the construction of the PSG based only on local criteria. It should be noted that if the termination condition depends on the ancestors of a node, the size of the resulting tree can be exponential in the number of rules. In contrast, the size of the query-tree may be exponential only in the arity of the predicates.

The Query-Tree in Knowledge Acquisition

A different use of the query-tree is as a tool for knowledge acquisition and knowledge base management. The query-tree essentially gives us a *view* (or *picture*) of the knowledge base relative to a query. It shows us exactly which derivations can be made and what formulas can be used in such derivations. A key problem in knowledge acquisition is that as the knowledge base grows, it is very hard to understand the interactions between the rules and the effects of changes. The query-tree shows us visually the dependence between rules and formulas. If a rule is removed, it can tell us that certain formulas have become irrelevant to the query, whereas if a rule is added, it can show us a dependency we have not anticipated. In that way it can also help us find erroneous rules (e.g., an over simplified or overly specific rule).

4.4 Summary

The query-tree is a very useful tool for many purposes. In this chapter we have explored only a few of its uses. We have shown that using the query-tree we can obtain significant speedups of inference, sometimes a few orders of magnitude. We have shown the query-tree to be a useful tool in deriving conclusions from external irrelevance claims and in extending other existing query evaluation methods.

Based on the observations made in this chapter, we can address one of the fundamental questions regarding the usage of explicit irrelevance claims. Namely, should we enable users to give a system explicit irrelevance claims that are based on additional knowledge that they may have, or should we require that they give the system the knowledge about the domain that underlies these irrelevance claims and develop methods for exploiting such knowledge to control inference. For example, instead of telling the system that flights are irrelevant to the query $Route(S.F., L.A., \$90)$, tell the system that:

- All flights cost more than \$100.
- Costs of busses and flights are all positive numbers.
- The sum of two positive numbers is positive, etc.

The system could then automatically derive that flights will be irrelevant to this query. The advantage of this approach[11] is that the underlying knowledge may be used in more flexible ways (e.g., it may be used for other queries as well). If the knowledge underlying the irrelevance claims changes, the system can automatically derive new irrelevance claims.

In general, this argument has much merit. When the knowledge underlying the irrelevance claims is available, there are clear advantages to giving a system that knowledge. In fact, the query-tree *is* a method for exploiting such knowledge effectively. However, there are several cases in which explicit irrelevance claims will be very useful:

1. It may not be possible to provide the knowledge underlying the irrelevance claims because of the expressive limits of the language being used. Going beyond the expressivity of the given language may affect the performance of the system significantly (even assuming it can support inference in more expressive languages). For example, stating that the join of two relations is empty cannot be done with Horn rules.

[11] Advocated to me by Matt Ginsberg

2. Providing the additional domain knowledge underlying an irrelevance claim may require adding a level of detail that is unwanted. For example, it may require adding new objects or predicates (in our example, the axioms of arithmetic), that will ultimately make the representation more complex and slow down inference.

3. The knowledge underlying the irrelevance claims may be of heuristic nature, and the user may not know the knowledge underlying it.

4. Irrelevance claims can be based on cached inferences made from the underlying domain knowledge. As long as their justifications are maintained, using them at run-time will lead to significant savings. The experimental results presented in this chapter can be viewed as a validation for this argument. The computation done by the query-tree and the indices created are actually precomputing irrelevance claims. Though these computations can be done at run-time, the experiments show that it is much less beneficial to do so.

5. As we see in Chapter 7, in some contexts we are given theories in which certain simplifying assumptions are made about the domain. In such cases, an explicit representation of these assumptions (via irrelevance claims) is useful in deciding when to use the given theories.

4.4.1 Related Work

In addition to the work described in the previous section, there are several other works related to the topics discussed in this chapter.

Building a query-tree and the corresponding indices for a query can be viewed as an instance of a general framework for knowledge compilation discussed in [Selman and Kautz, 1991]. In their framework, a new simpler knowledge base is created such that it will yield faster answers for a large number of the queries. For example, they show how to create *Horn approximations* of a theory that can be used in many cases to answer the query. One key difference between these approaches is that our transformed knowledge base is built with respect to a known set of queries, and for these queries inference will be more efficient.

A related approach is that of partial evaluation and in particular, partial evaluation of logic programs [Smith and Hickey, 1990; Lloyd and Shepherdson, 1991; Bruynooghe *et al.*, 1991]. Partial evaluation attempts to compile a set of rules in a way that will be efficient for a known set of queries. The query-tree method can be viewed as a generalization of previous methods for partial evaluation of constraint logic programs. In particular, work in logic programming has not emphasized the distinction between the rules in the program and the set of ground formulas, whereas

our approach argues that this distinction is necessary for our algorithms to be of practical interest. Moreover, the query-tree is the only partial evaluation procedure that yields the tightest constraints on the possible ground formulas that can appear in derivations of the query.

We have shown that the query-tree can be used to derive logical conclusions of irrelevance claims that are given to the system. A different approach, described in [Subramanian and Genesereth, 1987; Subramanian, 1989] is to give an axiomatization of irrelevance and use the axioms to reason about irrelevance claims. Conceivably, the same could be done in our framework by giving axiomatizations (or partial axiomatizations) to the various kinds of irrelevance claims in our space of definitions (in fact, Chapter 2 presents properties of irrelevance claims that can form a basis for such an axiomatization). However, we preferred to pursue the algorithmic approach since it is likely to be more efficient and its results are easier to characterize.

Chapter 5

Independence of Queries From Updates

This chapter considers the problem of detecting when a query is independent of an update to the knowledge base. This problem is primarily important because it enables us to save the computation needed to reevaluate a query after updates. Detecting independence is also a key issue in developing heterogeneous and distributed knowledge base systems [Genesereth, 1992; Litwin *et al.*, 1990]. In such systems, updates in one knowledge base may trigger updates in an other. For example, an important application that gives rise to such a setting is concurrent engineering [Cutkosky *et al.*, 1993; Levitt *et al.*, 1991], where several agents may be working on different parts of one design. Design decisions made by one agent may impose constraints on the possible design decisions of another agent, and therefore must be communicated. However, in order not to be burdened by excessive communication, only the changes that affect the other agents must be communicated. In database systems, detecting independence is important for several reasons. It can be used in order to maintain materialized views effectively.[1] In transaction scheduling we can provide greater flexibility by identifying that one transaction is independent of updates made by another. Finally, we can use independence in query optimization by ignoring parts of the database for which updates do not affect a specific query.

In this chapter we relate the independence problem to our framework for reasoning about irrelevance. We show that detecting independence is equivalent to detecting weak irrelevance. Making this connection sheds light on the independence problem, and enables us to significantly improve previous results in this area. In general, detecting independence is undecidable. However, viewing independence as a problem

[1] A view is a portion or an abstraction of a database. For example, a view may consist of an IDB relation. A materialized view is one that is maintained computed, as opposed to computing it on demand.

of detecting weak irrelevance yields new algorithms that provide sufficient conditions for independence by considering algorithms that provide sufficient conditions for detecting weak irrelevance. One such sufficient condition is strong irrelevance that we have examined in detail in Chapter 3.

A second sufficient condition for the case of Horn rule knowledge bases is based on the observation that detecting weak irrelevance can be couched as a problem of detecting equivalence of datalog programs. The notion of *uniform equivalence*, introduced in [Sagiv, 1988], can be used to provide a sufficient condition for equivalence of datalog programs. In order to use uniform equivalence for detecting independence, we extend the algorithm described in [Sagiv, 1988] to programs with interpreted literals and stratified negation. The result provides new decidable cases for independence and weak irrelevance and sound algorithms for the general case.

Our results significantly extend the known previous results on detecting independence. Specifically, it is shown that the results of [Blakeley *et al.*, 1989; Elkan, 1990] only capture strong irrelevance in datalog knowledge bases without recursion with additional restrictions on the rules. Our results extend the previous ones in two ways. We provide a strong irrelevance test to arbitrary datalog KBs (and the extensions described in Section 3.4), and we provide independence tests based on weak irrelevance which capture a larger class of independence than strong irrelevance.

5.1 Definitions

In this chapter we will consider knowledge bases containing a set of datalog (cf. [Ullman, 1989]) rules $\mathcal{P}$ and a set of ground formulas. We refer to the rules as a datalog *program*. We also allow the rules to have safe stratified negation. We denote the EDB predicates by $e_1, \ldots, e_n$ and the IDB predicates by $i_1, \ldots, i_n$. The input to a datalog program $\mathcal{P}$ is an EDB, i.e., a set of ground formulas for the EDB predicates, $E_1, \ldots, E_n$. We can also view $E_1, \ldots, E_n$ as relations for the EDB predicates in the intended interpretation of the knowledge base. A bottom-up evaluation is one in which we start with the ground EDB formulas and apply the rules to derive formulas for the IDB predicates. We continue applying the rules until no new formulas are generated.

We distinguish one IDB predicate as the *query* predicate. The *output* of the program $\mathcal{P}$ for the input $E_1, \ldots, E_m$, denoted $\mathcal{P}(E_1, \ldots, E_m)$, is the set of all ground formulas generated for the query predicate in the bottom-up evaluation. The query predicate is usually denoted as q.

A datalog program $\mathcal{P}$ is said to be *monotonic* (*anti-monotonic*) in the EDB predicates if for any input EDBs D_1 and D_2, $D_1 \supseteq D_2$ implies that $\mathcal{P}(D_1) \supseteq \mathcal{P}(D_2)$ ($\mathcal{P}(D_1) \subseteq \mathcal{P}(D_2)$). Containment and equivalence of datalog programs are defined as follows.

Definition 5.1: (Containment) A datalog program $\mathcal{P}_1$ *contains* a program $\mathcal{P}_2$, written $\mathcal{P}_2 \subseteq \mathcal{P}_1$, if for all EDBs $E_1, \ldots, E_m$, the output of $\mathcal{P}_1$ contains that of $\mathcal{P}_2$, i.e., $\mathcal{P}_2(E_1, \ldots, E_m) \subseteq \mathcal{P}_1(E_1, \ldots, E_m)$. ∎

Two programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are *equivalent*, written $\mathcal{P}_1 \equiv \mathcal{P}_2$, if $\mathcal{P}_2 \subseteq \mathcal{P}_1$ and $\mathcal{P}_1 \subseteq \mathcal{P}_2$. Containment of datalog programs is undecidable [Shmueli, 1987], even for programs without interpreted predicates or stratified negation. However, a weaker condition, *uniform containment*, was introduced and shown to be decidable in [Sagiv, 1988] for programs without interpreted predicates or stratified negation. In defining uniform containment, we assume that the input to a program $\mathcal{P}$ consists of relations for the EDB predicates $E_1, \ldots, E_m$ as well as *initial* relations for the IDB predicates $I_1^0, \ldots, I_n^0$.[2] The output of program $\mathcal{P}$ for $E_1, \ldots, E_m, I_1^0, \ldots, I_n^0$, written $\mathcal{P}(E_1, \ldots, E_m, I_1^0, \ldots, I_n^0)$, is computed as earlier by applying rules bottom-up until no new formulas are generated. When dealing with uniform containment (equivalence), we assume that the output is not just the relation for the query predicate but rather relations for all the IDB predicates $I_1, \ldots, I_n$ computed for the predicates $i_1, \ldots, i_n$, respectively. An output $I_1, \ldots, I_n$ *contains* another output $I_1', \ldots, I_n'$ if $I_j' \subseteq I_j$ $(1 \leq j \leq n)$.

Definition 5.2: (Uniform Containment) A program $\mathcal{P}_1$ *uniformly contains* $\mathcal{P}_2$, written $\mathcal{P}_2 \subseteq^u \mathcal{P}_1$, if for all EDBs $E_1, \ldots, E_m$ and for all initial IDBs $I_1^0, \ldots, I_n^0$,

$$\mathcal{P}_2(E_1, \ldots, E_m, I_1^0, \ldots, I_n^0) \subseteq \mathcal{P}_1(E_1, \ldots, E_m, I_1^0, \ldots, I_n^0).$$

∎

Two programs $\mathcal{P}_1$ and $\mathcal{P}_2$ are uniformly equivalent, written $\mathcal{P}_1 \equiv^u \mathcal{P}_2$, if $\mathcal{P}_2 \subseteq^u \mathcal{P}_1$ and $\mathcal{P}_1 \subseteq^u \mathcal{P}_2$. Uniform containment can also be explained in model-theoretic terms [Sagiv, 1988]. The uniform containment $\mathcal{P}_2 \subseteq^u \mathcal{P}_1$ holds if and only if $M(\mathcal{P}_1) \subseteq M(\mathcal{P}_2)$, where $M(\mathcal{P}_i)$ denotes the set of all models of $\mathcal{P}_i$.[3] Furthermore, for programs having only EDB predicates in bodies of rules, uniform containment is the same as containment. Note that a non-recursive program with no negation can be transformed into this form by unfolding of the rules. Consequently, uniform equivalence provides a necessary condition for equivalence for non-recursive programs.

[2] Note that in defining equivalence, the initial relations for the IDB predicates are assumed to be empty.

[3] Note that, in contrast, containment holds if the set of *minimal fixed-point* models of $\mathcal{P}_1$ are contained in those of $\mathcal{P}_2$.

5.1.1 Updates and Independence

Given a datalog program $\mathcal{P}$, which we call the *query program*, we consider updates to the EDB predicates of $\mathcal{P}$, denoted by $e_1, \ldots, e_m$. In an update, we either remove or add ground formulas to the extensional database. To simplify notation, we assume that updates are always done on the relation E_1 for e_1. To specify the set of formulas that are updated in E_1, we assume we have another datalog program, called the *update program*, denoted by $\mathcal{P}^u$. The query predicate of $\mathcal{P}^u$ is u, and its arity is equal to that of e_1. The tuples computed for u will be the set of tuples updated in E_1.

We assume (without loss of generality) that the IDB predicates of $\mathcal{P}^u$ are different from those of $\mathcal{P}$. The EDB predicates of $\mathcal{P}^u$, however, could be EDB predicates of $\mathcal{P}$, as well as predicates not appearing in $\mathcal{P}$. To distinguish the two sets of EDB predicates, we will use the phrase "EDB predicates" to refer exclusively to the EDB predicates $e_1, \ldots, e_m$ of the query program $\mathcal{P}$; the other extensional predicates that may appear in the update program are referred to as *base predicates*, denoted by $b_1, \ldots, b_l$. Of course, some of the EDB predicates may also appear in the update program $\mathcal{P}^u$. We denote the output of update program $\mathcal{P}^u$ as $\mathcal{P}^u(E_1, \ldots, E_m, B_1, \ldots, B_l)$, even if $\mathcal{P}^u$ does not use all (or any) of the EDB predicates. Sometimes we refer to the output of $\mathcal{P}^u$ simply by U.

An update is either an *insertion* or a *deletion* and it applies to the relation E_1 for the EDB predicate e_1. The tuples to be inserted into or deleted from E_1 are those in the relation computed for u. A large class of updates consists of those *not* depending on the EDB relations, as captured by the following definition:

Definition 5.3: (Oblivious Update) An update specified by an update program $\mathcal{P}^u$ is *oblivious* with respect to a query program $\mathcal{P}$ if $\mathcal{P}^u$ has only base predicates (and no EDB predicates). An update is *nonoblivious* if the update program $\mathcal{P}^u$ has some EDB predicates (and possibly some base predicates). ∎

To define independence, suppose we are given a query program $\mathcal{P}$ and an update program $\mathcal{P}^u$. The program $\mathcal{P}$ is *independent* of the given update if the update does not change the answer to the query predicate. The formal definition is as follows.

Definition 5.4: (Independence): A program $\mathcal{P}$ is independent of an update specified by a program $\mathcal{P}^u$ if for all EDB relations $E_1, \ldots, E_m$ and for all base relations $B_1, \ldots, B_l$,

$$\mathcal{P}(E_1, E_2, \ldots, E_n) = \mathcal{P}(E_1', E_2, \ldots, E_n),$$

where E_1' is the result of applying the update to E_1; that is, $E_1' = E_1 \cup U$ if the update is an insertion, and $E_1' = E_1 - U$ if the update is a deletion, where $U = \mathcal{P}^u(E_1, \ldots, E_m, B_1, \ldots, B_l)$. ∎

We use the following notation. $In^+(\mathcal{P}, \mathcal{P}^u)$ means that program $\mathcal{P}$ is independent of the insertion specified by the update program $\mathcal{P}^u$. Similarly, $In^-(\mathcal{P}, \mathcal{P}^u)$ means that program $\mathcal{P}$ is independent of the deletion specified by the update program $\mathcal{P}^u$.

Example 5.5: Consider the following program P_1:

$inCar(X, Y, A) \wedge driver(X) \wedge inCar(Z, Y, B) \wedge B \geq 18 \Rightarrow canDrive(X, Y, A)$
$canDrive(X, Y, A) \wedge A \geq 18 \Rightarrow adultDriver(X)$

An atom $canDrive(X, Y, A)$ is true if person X can drive car Y and A is the age of X. According to the rule for $canDrive$, person X can drive car Y if X is a driver and there is someone of the age 18 or older in the same car. An adult driver, as computed by the IDB predicate $adultDriver$, is anyone who can drive a car and is of the age 18 or older. Let the update program $\mathcal{P}_1^u$ consist of the rule:

$inCar(X, Y, A) \wedge \neg driver(X) \wedge A < 18 \Rightarrow u_1(X, Y, A)$

and suppose that the deletion defined by u_1 is applied to $inCar$; that is, non-drivers under the age of 18 are removed from $inCar$. The query predicate $adultDriver$ is independent of the deletion update $\mathcal{P}_1^u$ because the existence of non-drivers under the age of 18 does not affect the ability to derive that a person can drive. ∎

Several properties of independence are shown by Elkan [Elkan, 1990]. In particular, he showed the following.

Lemma 5.6: *Consider a query program $\mathcal{P}$ and an update program $\mathcal{P}^u$. If $\mathcal{P}^u$ is monotonic in the EDB predicates, and $\mathcal{P}$ is either monotonic or anti-monotonic in the EDB predicates, then*

$$In^-(\mathcal{P}, \mathcal{P}^u) \Longrightarrow In^+(\mathcal{P}, \mathcal{P}^u).$$

Similarly to the above lemma, we can also prove the following.

Lemma 5.7: *Consider a query program $\mathcal{P}$ and an update program $\mathcal{P}^u$. If $\mathcal{P}^u$ is anti-monotonic in the EDB predicates, and $\mathcal{P}$ is either monotonic or anti-monotonic in the EDB predicates, then*

$$In^+(\mathcal{P}, \mathcal{P}^u) \Longrightarrow In^-(\mathcal{P}, \mathcal{P}^u). \quad -$$

Proof: Consider an EDB $E_1, \ldots, E_n$, denoted as $\bar{E}$, and relations $B_1, \ldots, B_m$, denoted as $\bar{B}$, for the base predicates. The tuples of the update are given by $U = \mathcal{P}^u(\bar{E}, \bar{B})$. A deletion update transforms the EDB $\bar{E}$ into the EDB $E_1 - U, \ldots, E_n$, denoted as $\bar{E}^-$. We have to show the following:

$$\mathcal{P}(\bar{E}^-) = \mathcal{P}(\bar{E}).$$

Consider the EDB $\bar{E}^-$ with the relations $\bar{B}$ for the base predicates. Let $U'' = \mathcal{P}^u(\bar{E}^-, \bar{B})$. Since $\mathcal{P}^u$ is anti-monotonic in the EDB, $U' \subseteq U''$.

We now apply the insertion update specified by $U'' = \mathcal{P}^u(\bar{E}^-, B)$ to $\bar{E}^-$, yielding

$$(E_1 - U') \cup U'', E_2, \ldots, E_n.$$

Since $In^+(\mathcal{P}, \mathcal{P}^u)$ is assumed, we get

$$\mathcal{P}(E^-) = \mathcal{P}((E_1 - U') \cup U'', E_2, \ldots, E_n). \tag{5.1}$$

Moreover, $U' \subseteq U''$ implies

$$E_1 - U' \subseteq E_1 \subseteq (E_1 - U') \cup U''. \tag{5.2}$$

If $\mathcal{P}$ is monotonic in the EDB, then (5.2) implies

$$\mathcal{P}(\bar{E}^-) \subseteq \mathcal{P}(E) \subseteq \mathcal{P}((E_1 - U') \cup U'', E_2, \ldots, E_n),$$

and, so, from (5.1) we get

$$\mathcal{P}(\bar{E}^-) = \mathcal{P}(\bar{E}).$$

Similarly, if $\mathcal{P}$ is anti-monotonic in the EDB, then (5.2) implies

$$\mathcal{P}(\bar{E}^-) \supseteq \mathcal{P}(\bar{E}) \supseteq \mathcal{P}((E_1 - U) \cup U'', E_2, \ldots, E_n),$$

and, so, from (5.1) we get

$$\mathcal{P}(\bar{E}^-) = \mathcal{P}(\bar{E}).$$

■

Note that if an update is oblivious, then it is both monotonic and anti-monotonic. Therefore, the above two lemmas imply the following corollary.

Corollary 5.8: *Consider a query program $\mathcal{P}$ and an update program $\mathcal{P}^u$. If the update is oblivious (i.e., EDB predicates of $\mathcal{P}$ do not appear in $\mathcal{P}^u$), and $\mathcal{P}$ is either monotonic or anti-monotonic in the updated EDB predicates, then the following equivalence holds:*

$$In^-(\mathcal{P}, \mathcal{P}^u) \Longleftrightarrow In^+(\mathcal{P}, \mathcal{P}^u).$$

The importance of Lemma 5.7 and Corollary 5.8, as we will see in the next section, lies in the fact that testing $In^+(\mathcal{P}, \mathcal{P}^u)$ is usually easier than testing $In^-(\mathcal{P}, \mathcal{P}^u)$.

5.2 Irrelevance, Independence and Equivalence

In this section, we formalize the connection between three related problems: irrelevance of formulas and independence and equivalence of datalog programs. We show that independence of a deletion update is equivalent to a form of weak irrelevance. We also show that the independence problem can be formulated as a problem of equivalence of two datalog programs. We exploit this connection as follows:

- We develop algorithms that provide a sufficient condition for independence based on strong irrelevance.
- We develop novel algorithms for detecting equivalence of datalog programs. As a result, we get algorithms for detecting independence and weak irrelevance.

Irrelevance and Independence

As stated, independence of a deletion update is equivalent to weak irrelevance. Given an update program $\mathcal{P}^u$ and a database D (consisting the EDB relations and the base relations), we denote by $E_1^{U'}(D)$ all the formulas of the form $e_1(a_1, \ldots, a_n)$, where $u(a_1, \ldots, a_n) \in \mathcal{P}^u(D)$.

Lemma 5.9: *Let $\mathcal{P}$ be a datalog program with query predicate q and $\mathcal{P}^u$ be an update program, where both $\mathcal{P}$ and $\mathcal{P}^u$ have no negation. The independence $In^-(\mathcal{P}, \mathcal{P}^u)$ holds if and only if for any database D, $WI(E_1^{U'}(D), q, \mathcal{P} \cup D, DI_1, \mathcal{D}_q)$ holds.*

Proof: Assume that $In^-(\mathcal{P}, \mathcal{P}^u)$ holds, and let $D = E_1, \ldots, E_n, B_1, \ldots, B_m$ be an arbitrary database and let $q(\bar{a}) \in \mathcal{P}(D)$. To show weak irrelevance, we must show that $q(\bar{a})$ has a derivation that does not use formulas in $E_1^{U'}$. However, since $In^-(\mathcal{P}, \mathcal{P}^u)$ holds,

$$\mathcal{P}(E_1, \ldots, E_n) = \mathcal{P}(E_1 - U', E_2, \ldots, E_n),$$

where $U' = \mathcal{P}^u(D)$. This means that $q(\bar{a})$ has a derivation d from $E_1 - U', E_2, \ldots, E_n$, and d does not contain formulas in $E_1^{U'}$.

Conversely, suppose that $WI(E_1^{U'}(D), q, \mathcal{P} \cup D, DI_1, \mathcal{D}_q)$ holds for any database D. Clearly,

$$\mathcal{P}(E_1, \ldots, E_n) \supseteq \mathcal{P}(E_1 - U', \ldots, E_n).$$

To show independence, we need to show that

$$\mathcal{P}(E_1, \ldots, E_n) \subseteq \mathcal{P}(E_1 - U', \ldots, E_n).$$

Let $q(\bar{a}) \in \mathcal{P}(E_1, \ldots, E_n)$. Since $E_1^{U'}$ is weakly irrelevant to q, $q(\bar{a})$ will have a derivation d that does not contain formulas in $E_1^{U'}$. Therefore, d will also be a derivation of

$q(\bar{a})$ from $\mathcal{P} \cup E_1 - U', E_2, \ldots, E_n$, and so $q(\bar{a}) \in \mathcal{P}(E_1 - U', \ldots, E_n)$ and the independence holds. ∎

Elkan [Elkan, 1990] shows that detecting independence in general is undecidable. This also follows from Lemma 5.9, since weak irrelevance is undecidable in general. However, viewing independence as weak irrelevance provides insight into the problem of detecting independence. For example, the following corollary will yield an algorithm for detecting independence.

Corollary 5.10: *If for any database D, $SI(E_1^{U'}(D), q, \mathcal{P} \cup D, DI_1, \mathcal{D}_q)$ holds, then $In^-(\mathcal{P}, \mathcal{P}^u)$.*

The corollary follows from Lemma 5.9 and Lemma 2.7. To detect strong irrelevance of $E_1^{U'}$, we create a new datalog program that explicitly contains a predicate representing the relation $E_1^{U'}$. Specifically, given the rules $\mathcal{P}$ and $\mathcal{P}^u$, we create a new program $\mathcal{P}_1$ as follows:

1. $\mathcal{P}_1$ includes the rules of $\mathcal{P}$ and $\mathcal{P}^u$.
2. $\mathcal{P}_1$ includes the rule
$$e_1(\bar{X}) \wedge u(\bar{X}) \Rightarrow e_1^u(\bar{X})$$
that defines the relation $E_1^{U'}$.
3. $\mathcal{P}_1$ includes rules that enable using formulas of e_1^u, whenever the corresponding formulas of e_1 would be used. Specifically, let $e_1(\bar{X})$ be some occurrence of e_1 in a rule $r \in \mathcal{P}$. The program $\mathcal{P}_1$ includes the rule r' created by replacing the literal $e_1(\bar{X})$ in the antecedent of r by $e_1^u(\bar{X})$.

The following lemma assures that detecting strong irrelevance of formulas of e_1^u in $\mathcal{P}_1$ will entail independence of $\mathcal{P}^u$:

Lemma 5.11: *Let D be a database and $e_1(\bar{a})$ be a formula such that $u(\bar{a}) \in \mathcal{P}^u(D)$. Then $e_1(\bar{a})$ is part of a derivation of the query from $\mathcal{P} \cup D$ if and only if $e_1^u(\bar{a})$ is part of a derivation of the query from $\mathcal{P}_1 \cup D$.*

Proof: Let d be a derivation from $\mathcal{P} \cup D$ that contains $e_1(\bar{a})$; i.e., d uses an instance of a rule r of the form
$$e_1(a) \wedge s_1(\bar{a}_1) \wedge \ldots \wedge s_l(\bar{a}_l) \Rightarrow p(\bar{b}).$$
We can assume that the leftmost literal in the antecedent is $e_1(\bar{a})$. Recall that $\bar{a} \in E_1^{U'}$, and therefore, there is a derivation d_u of $u(\bar{a})$ from $\mathcal{P}_1 \cup D$ (using only the rules coming from $\mathcal{P}^u$). The program $\mathcal{P}_1$ contains a rule r' in which the leftmost literal in

the antecedent of r, $e_1(\bar{X})$, is replaced by $e_1^u(\bar{X})$. To create a derivation of the query d' that uses $e_1^u(\bar{a})$, we replace r by r'. The goal-node $e_1^u(\bar{a})$ is unified with the rule defining e_1^u, resulting in the instantiated rule

$$e_1(\bar{a}) \wedge u(\bar{a}) \Rightarrow e_1^u(\bar{a}).$$

The atom $e_1(\bar{a})$ is obviously satisfied (because it was satisfied in d). To complete d', we make the atom $u(\bar{a})$ the head of the derivation d_u.

For the other direction, let d' be a derivation from $\mathcal{P}_1 \cup D$ that contains $e_1^u(\bar{a})$. We simply reverse the transformation we performed above and get a derivation d' of the query from $\mathcal{P} \cup D$. ∎

Consequently, if there is no node of e_1^u in the query-tree of $\mathcal{P}_1$, then $E_1^U(D)$ is strongly irrelevant to the query, and therefore, by Corollary 5.10, $In^-(\mathcal{P}, \mathcal{P}^u)$ holds.

Example 5.12: Consider again our *goodPath* example given by the rules:

$r_1 : badPoint(X) \wedge path(X, Y) \wedge goodPoint(Y) \Rightarrow goodPath(X, Y)$
$r_2 : link(X, Y) \Rightarrow path(X, Y)$
$r_3 : link(X, Z) \wedge path(Z, Y) \Rightarrow path(X, Y)$
$r_4 : step(X, Y) \Rightarrow link(X, Y)$

and the additional constraints:

$badPoint(X) \Rightarrow 100 < X < 200$
$step(X, Y) \Rightarrow X < Y$
$goodPoint(X) \Rightarrow 150 < X < 170.$

Suppose we want to remove the formulas of $step(X, Y)$ for which $X < 90$. We would add the following rules to the program:

$r_5 : step(X, Y) \wedge (X < 90) \Rightarrow lowStep(X, Y)$
$r_6 : lowStep(X, Y) \Rightarrow link(X, Y).$

The query-tree built for *goodPath* will be identical to the one shown in Figure 4.1 and will not contain a node of the predicate *lowStep*. Therefore, *goodPath* is independent of this update. ∎

Independence and Equivalence

The independence problem can also be formulated as a problem of detecting equivalence of datalog programs. To show that, we construct a new program that computes the new value of the query predicate q (after the update) from the old value of the EDB (before the update). One program, $\mathcal{P}^+$, is constructed for the case of insertion,

and another program, $\mathcal{P}^-$, is constructed for the case of deletion. We then pose the independence problem as the equivalence of the original program and $\mathcal{P}^+$ (or $\mathcal{P}^-$). Each of $\mathcal{P}^+$ and $\mathcal{P}^-$ consists of three parts:

- The rules of $\mathcal{P}$, after all occurrences of the predicate name e_1 have been replaced by a new predicate name s.
- The rules of the update program $\mathcal{P}^u$.
- Rules for the new predicate s.

$\mathcal{P}^+$ and $\mathcal{P}^-$ differ only in the third part. In the case of insertion, the predicate s in $\mathcal{P}^+$ is intended to represent the relation E_1 after the update, and therefore the rules for s are:

$$e_1(X_1, \ldots, X_k) \Rightarrow s(X_1, \ldots, X_k)$$
$$u(X_1, \ldots, X_k) \Rightarrow s(X_1, \ldots, X_k)$$

In the case of deletion, the predicate s in $\mathcal{P}^-$ is intended to represent the deletion update to E_1, and the rule for defining it is

$$e_1(X_1, \ldots, X_k) \wedge \neg u(X_1, \ldots, X_k) \Rightarrow s(X_1, \ldots, X_k).$$

Note that since $\mathcal{P}$ and $\mathcal{P}^u$ do not share IDB predicates, the negation in the $\mathcal{P}^-$ will be stratified. The following propositions are immediate corollaries of the definition of independence.

Proposition 5.13: $In^+(\mathcal{P}, \mathcal{P}^u) \Longleftrightarrow \mathcal{P} \equiv \mathcal{P}^+$.

Proposition 5.14: $In^-(\mathcal{P}, \mathcal{P}^u) \Longleftrightarrow \mathcal{P} \equiv \mathcal{P}^-$.

Proof: Both propositions follow from the observation that the relation computed for s is the updated relation for E_1. Therefore, since e_1 is replaced by s in the rules of the program, the new program will compute the relation for q after the update. Clearly, the independence holds if and only if the new program is equivalent to the original program. ∎

Returning to Example 5.5, both $\mathcal{P}^+$ and $\mathcal{P}^-$ will have the following rules:

$s(X, Y, A) \wedge driver(X) \wedge s(Z, Y, B) \wedge B \geq 18 \Rightarrow canDrive(X, Y, A)$
$canDrive(X, Y, A) \wedge A \geq 18 \Rightarrow adultDriver(X)$
$inCar(X, Y, A) \wedge \neg driver(X) \wedge A < 18 \Rightarrow u_1(X, Y, A)$.

The program $\mathcal{P}^+$ will contain the rules:

$inCar(X,Y,A) \Rightarrow s(X,Y,A)$
$u_1(X,Y,A) \Rightarrow s(X,Y,A)$

and the program $\mathcal{P}^-$ will contain the rule:

$inCar(X,Y,A) \wedge \neg u_1(X,Y,A) \Rightarrow s(X,Y,A)$.

Both $\mathcal{P}^+$ and $\mathcal{P}^-$ are equivalent to the original program, and therefore, the query $adultDriver(X)$ is independent of the insertion and deletion updates of $\mathcal{P}^u$.

5.2.1 Understanding Previous Work

Relating the independence problem to irrelevance enables us to understand better previous work on independence by [Blakeley *et al.*, 1989] and [Elkan, 1990]. Both of them considered restricted languages in which weak irrelevance is the same as strong irrelevance. The result of Blakeley et al. [Blakeley *et al.*, 1989] applies just to conjunctive queries (cf. [Ullman, 1989]), i.e., knowledge bases in which the antecedents of every rule are EDB predicates. Furthermore, the rules are restricted such that every predicate can only appear once in the antecedent. Elkan generalizes the result by Blakeley et al. to deal with interpreted constraints and only requires that the query be *conjunctive* in the updated predicate, as defined below. In the definition, $Def(q)$ denotes all the predicates that can appear in a derivation of q:

Definition 5.15: A query q is conjunctive in the updated predicate e_1 if it is defined by a single rule of the form:

$$e_1(\bar{Y}) \wedge s_1(\bar{X}_1) \wedge \ldots \wedge s_n(\bar{X}_n) \Rightarrow q(\bar{X}),$$

where e_1 has a single appearance in the rule, and $e_1 \notin Def(s_i)$ for $1 \leq i \leq n$. ∎

Under this restriction, weak irrelevance is equivalent to strong irrelevance:

Observation 5.16: Let the query q be conjunctive in the predicate e_1, then for any formula $e_1(\bar{b})$, $WI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, DI_1, \mathcal{D}_q) \iff SI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, DI_1, \mathcal{D}_q)$.

Proof: We only need to show that $WI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, DI_1, \mathcal{D}_q) \Longrightarrow SI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, \mathcal{D}_q)$. Assume by contradiction that $WI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, DI_1, \mathcal{D}_q)$ holds, and suppose D is a database in which $e_1(\bar{b})$ is used in a derivation d of $q(\bar{a})$. Consider the database D' which is identical to D except that the relation E_1 includes *only* the tuple $\bar{b}$. The derivation d is still a valid derivation of $q(\bar{a})$ from D', since e_1 is only used once in d. However, there is no derivation of $q(\bar{a})$ from D' that does not use $e_1(\bar{b})$, because some formula of e_1 must be used in the derivation of $q(\bar{a})$. Consequently, $WI(e_1(\bar{b}), q, \Sigma_{\mathcal{P}}, DI_1, \mathcal{D}_q)$ cannot hold. ∎

Using the query-tree to detect independence is an immediate generalization of Elkan's algorithm. It provides a strong irrelevance test for arbitrary datalog programs, thereby removing the restrictions that the program cannot have recursion and that the query must be conjunctive in the updated predicate.

In the next section, we describe algorithms for deciding equivalence of datalog programs and therefore for detecting weak irrelevance. The results provide new decidable cases for independence and provide sufficient conditions for independence in the general case.

To illustrate the added power of our algorithm, consider the query $adultDriver(X)$ when the update program is:

$$inCar(X, Y, A) \wedge \neg driver(X) \Rightarrow u_2(X, Y, A).$$

Suppose we apply the deletion defined by u_2 to the relation $inCar$, i.e., we remove any person who is not a driver. The query $adultDriver$ is independent of this deletion update (because X and Z can be bound to the same constant in the rule defining $canDrive$). However, there are derivations of $adultDriver(X)$ that use adult non-drivers, and therefore the formulas computed by u_2 are not strongly irrelevant to the query $adultDriver$. Consequently, Elkan's algorithm will not detect the independence in this example.

5.3 Testing Equivalence of Datalog Programs

In the remainder of this chapter we consider the problem of testing equivalence of datalog programs. As stated earlier, solutions to this problem directly impact the independence problem. Shmueli [Shmueli, 1987] showed that detecting equivalence of two datalog programs is in general undecidable even if the programs do not contain interpreted predicates or negation. Sagiv [Sagiv, 1988] introduced a weaker condition, uniform equivalence, and showed that it is decidable for datalog programs without interpreted predicates or negation. Recall that the reduction of the problem of independence to equivalence involved testing equivalence of programs with stratified negation. Therefore, in order to use uniform equivalence for detecting independence, we extend the algorithms described in [Sagiv, 1988] to handle both interpreted predicates (which are defined in Section 2.4) and stratified negation. Specifically, we show the following:

Theorem 5.17: *Testing whether a datalog program $\mathcal{P}_1$ is uniformly equivalent to a datalog program $\mathcal{P}_2$ is decidable even if $\mathcal{P}_1$ and $\mathcal{P}_2$ include interpreted predicates and stratified negation.*

As a consequence of this theorem, we get the following results for testing independence:

Corollary 5.18: *Independence is decidable in the following cases:*

1. *$In^+(\mathcal{P}, \mathcal{P}^u)$ ($In^-(\mathcal{P}, \mathcal{P}^u)$) is decidable if both $\mathcal{P}^+$ ($\mathcal{P}^-$) and $\mathcal{P}$ have only interpreted and EDB predicates (that may appear positively or negatively) in bodies of rules.*[4]

2. *Both $In^+(\mathcal{P}, \mathcal{P}^u)$ and $In^-(\mathcal{P}, \mathcal{P}^u)$ are decidable if $\mathcal{P}$ is non-recursive, and $\mathcal{P}^u$ has only rules of the form*

$$e_1(X_1, \ldots, X_k) \wedge c \Rightarrow u(X_1, \ldots, X_k),$$

where c is a conjunction of interpreted literals such that $\neg c$ is expressible in the constraint language.

Proof: The first half of the corollary follows from the observation that for these classes of programs, uniform equivalence is also a necessary condition for equivalence. The second half holds because in both $\mathcal{P}^+$ and $\mathcal{P}^-$ the rules defining s will not contain negation and therefore both $\mathcal{P}^+$ and $\mathcal{P}^-$ can be rewritten equivalently to satisfy the conditions of the first half of the corollary. ∎

Corollary 5.19:

1. *$In^+(\mathcal{P}, \mathcal{P}^u)$ is decidable if both $\mathcal{P}$ and $\mathcal{P}^u$ are non-recursive and only EDB predicates appear negated.*

2. *If, in addition, the update is oblivious, then $In^-(\mathcal{P}, \mathcal{P}^u)$ is decidable.*

Proof: The first half follows from the observation that under the conditions stated, both $\mathcal{P}$ and $\mathcal{P}^+$ can be rewritten to satisfy the condition of Corollary 5.18. The second half follows from the first and from Corollary 5.8. ∎

5.3.1 Uniform Equivalence with Interpreted Predicates

The algorithm for detecting uniform containment (and equivalence) for datalog programs without interpreted predicates is based on the model theoretic characterization of the notion, shown in [Sagiv, 1988], which also holds for programs with interpreted predicates. Specifically, the uniform containment $\mathcal{P}_2 \subseteq^u \mathcal{P}_1$ holds if and only if $M(\mathcal{P}_1) \subseteq M(\mathcal{P}_2)$, where $M(\mathcal{P}_i)$ denotes the set of all models of $\mathcal{P}_i$.[5] We note that

[4] We prefer to describe this case in terms of $\bar{\mathcal{P}}$ and $\mathcal{P}^+$, rather than $\mathcal{P}$ and $\mathcal{P}^u$, since it is clearer.

[5] Note that when we consider the case of interpreted predicates, the models of $\mathcal{P}_i$ must map the interpreted predicates to their natural interpretation.

$M(\mathcal{P}_1) \subseteq M(\mathcal{P}_2)$ holds if and only if $M(\mathcal{P}_1) \subseteq M(r)$ for every rule $r \in \mathcal{P}_2$, since a database D is a model of $\mathcal{P}_2$ if and only if it is a model of every rule $r \in \mathcal{P}_2$. Therefore, we can decide whether $M(\mathcal{P}_1) \subseteq M(\mathcal{P}_2)$ by checking whether $M(\mathcal{P}_1) \subseteq M(r)$ for every $r \in \mathcal{P}_2$.

Based on this observation, when the programs have no interpreted predicates, the following algorithm (from [Sagiv, 1988]) will decide whether a given rule r is uniformly contained in a program $\mathcal{P}$. Given a rule r of the form

$$q_1 \wedge \ldots \wedge q_n \Rightarrow p,$$

we use a substitution θ that maps every variable in the antecedent of r to a distinct symbol that does not appear in $\mathcal{P}$ or r. We then apply the program $\mathcal{P}$ to the atoms $q_1\theta, \ldots, q_n\theta$. Sagiv shows that the program $\mathcal{P}$ generates $p\theta$ from $q_1\theta, \ldots, q_n\theta$ if and only if $M(\mathcal{P}) \subseteq M(r)$.

Example 5.20: Suppose we are trying to determine whether the rule

$$r_1 : e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$$

is contained in the program P_1:

$$p(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$$
$$e(X, Y) \Rightarrow p(X, Y).$$

We apply P_1 to the ground atoms $e(x_0, z_0)$ and $p(z_0, y_0)$. Since we are able to derive $p(x_0, y_0)$, the rule r_1 is contained in P_1. ∎

However, there is a problem in applying this algorithm to programs with interpreted predicates. First, the constants used in the input to $\mathcal{P}$, i.e., those that appear in $q_1\theta, \ldots, q_n\theta$, are arbitrary, and therefore, interpreted predicates are not defined on them. Consequently, the interpreted literals in the rules (that may involve $<, \leq$, etc.) cannot be evaluated. Moreover, some of the derivations of $p\theta$ by $\mathcal{P}$ depend on the symbols satisfying the interpreted constraints, and so these cannot be discarded.

We address this problem by associating a constraint with every fact involved in the evaluation of $\mathcal{P}$. The constraints for a given fact f represent the conditions on the constants in $q_1\theta, \ldots, q_n\theta$ under which f is derivable. We manipulate these constraints as we evaluate $\mathcal{P}$. Formally, let r be the rule

$$q_1 \wedge \ldots \wedge q_n \wedge c_r \Rightarrow p. \tag{5.3}$$

We denote the set of variables in r by Y. The subgoal c_r is the conjunction of the literals of interpreted predicates in r. We assume that all literals in r have distinct variables in every argument position. Note that this requirement can always be

fulfilled by introducing additional literals using the $=$ predicate. As in the original algorithm, we define a mapping θ that maps each variable in r to a distinct symbol not appearing in $\mathcal{P}$ or r. Instead of evaluating $\mathcal{P}$ with the ground atoms $q_1\theta, \ldots, q_n\theta$, we evaluate $\mathcal{P}$ with facts that are pairs of the form (q, c), where q is a ground atom and c is a constraint on the symbols in $Y\theta$. The input to $\mathcal{P}$ will be the pairs $(q_i\theta, c_r\theta)$, for $i = 1, 2, \ldots, n$.

An application of a rule

$$g_1 \wedge \ldots \wedge g_l \wedge c \Rightarrow h$$

proceeds as follows. Let $(a_1, c^1), \ldots, (a_l, c^l)$ be pairs generated previously, such that there is a substitution τ for which $g_i\tau = a_i$ $(1 \leq i \leq l)$. Let c_h be the conjunction $c^1 \wedge \ldots \wedge c^l \wedge c\tau$. If c_h is satisfiable, we derive the pair $(h\tau, c_h)$. In words, the constraint of the new pair generated is the conjunction of the constraints on the pairs used in the derivation and the constraints of the rule that was applied in that derivation. We apply the rules of $\mathcal{P}$ until no new pairs are generated. Note that there are only a finite number of possible constraints for the generated pairs and, therefore, the bottom-up evaluation must terminate.

Finally, let $(p\theta, c_1), \ldots, (p\theta, c_m)$ be all the pairs generated for the atom $p\theta$ in the evaluation of $\mathcal{P}$; recall that p is the head of Rule (5.3) and θ is the substitution used to convert the variables of that rule to new symbols. As we will prove, the containment $M(\mathcal{P}) \subseteq M(r)$ holds if and only if $c_r \models c_1 \vee \ldots \vee c_m$, where c_r is the conjunction of interpreted predicates from the antecedent of Rule (5.3).

Example 5.21: Let $\mathcal{P}_1$ be the program:

$r_1 : e(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$
$r_2 : e(X, Y) \Rightarrow q(X, Y)$.

Let $\mathcal{P}_2$ be the program:

$s_1 : p(X, Z) \wedge p(Z, Y) \Rightarrow p(X, Y)$
$s_2 : e(X, Y) \wedge (X \leq Y) \Rightarrow p(X, Y)$
$s_3 : e(X, Y) \wedge (Y \leq X) \Rightarrow q(X, Y)$
$s_4 : p(X, Y) \Rightarrow q(X, Y)$.

For a variable X of a rule r, we denote the constant $X\theta$ by x_0. $True$ denotes the constraint satisfied by all tuples. To check the uniform containment of r_1 in $\mathcal{P}_2$, the input to $\mathcal{P}_2$ would be $(e(x_0, z_0), True)$ and $(p(z_0, y_0), True)$. Rule s_2 will derive $(p(x_0, z_0), x_0 \leq z_0)$ and rule s_1 will then derive $(p(x_0, y_0), x_0 \leq z_0)$. Since $p(x_0, y_0)$ was only generated under the constraint $x_0 \leq z_0$, the rule r_1 is not uniformly contained in $\mathcal{P}_2$.

To check the uniform containment of rule r_2 in $\mathcal{P}_2$, we begin with $(e(x_0, y_0), True)$. Rule s_3 will then derive $(q(x_0, y_0), y_0 \leq x_0)$. Rule s_2 will derive $(p(x_0, y_0), x_0 \leq y_0)$,

and rule s_4 will use that to derive $(q(x_0, y_0), x_0 \leq y_0)$. Since $q(x_0, y_0)$ was derived for both possible orderings of x_0 and y_0, rule r_2 is uniformly contained in $\mathcal{P}_2$. However, since r_1 is not uniformly contained in $\mathcal{P}_2$, the program $\mathcal{P}_1$ is not contained in $\mathcal{P}_2$. ∎

To prove the correctness of the algorithm, the following lemma relates derivations of pairs to those of ground atoms.

Lemma 5.22: *Let d be a derivation of the pair $(p\theta, c)$ from $\mathcal{P}$ and the database containing the pairs $(q_1\theta, c_r\theta), \ldots, (q_n\theta, c_r\theta)$, and let τ be a substitution that maps each variable of r to a constant such that the constraint c is satisfied by $Y\tau$. Let d' be the derivation in which every node $(n\theta, c_0\theta)$ in d is replaced by the ground formula $n\tau$. The derivation d' is a valid derivation of $p\tau$ from $\mathcal{P}$ and $Y\tau$.*

Proof: To prove the lemma, we need to show that in every rule application in d', the constants that are involved satisfy the interpreted constraints. The proof is based on the following observation. The constraint c_0 in a node $(n\theta, c_0\theta)$ in d is stronger than the constraints of its subgoals and stronger than the conjunction of interpreted literals in the rule used to derive $(n\theta, c_0\theta)$. This follows from the way we evaluated $\mathcal{P}$ with pairs, where the constraint of the head pair was the conjunction of the constraints of the rule being applied and the subgoals used. Therefore, since $Y\tau$ satisfies the constraint in the root of d, then it satisfies the constraints of all the nodes in d. ∎

Based on this lemma, the correctness of the algorithm is established by the following theorem.

Theorem 5.23: $M(P) \subseteq M(r) \iff c_r \models c_1 \vee \ldots \vee c_m$.

Proof: For the first direction, assume $c_r \models c_1 \vee \ldots \vee c_m$. We need to show that $M(P) \subseteq M(r)$. To show that, it is enough to show the following. If τ is a substitution that maps each variable in Y to a constant such that

1. $Y\tau$ satisfies c_r, and
2. $q_1\tau, \ldots, q_n\tau \in M$, where $M \in M(P)$,

then $p\tau \in M$. We show that by showing that if P is applied to the inputs $q_1\tau, \ldots, q_n\tau$, then $p\tau$ will be derived.

Since $Y\tau$ satisfies c_r there exists at least one i, $1 \leq i \leq m$, such that $Y\tau$ satisfies c_i. Consider the derivation of $(p\theta, c_i)$ from P. Lemma 5.22 guarantees that $p\tau$ has a derivation from $q_1\tau, \ldots, q_n\tau$.

For the second direction, suppose $c_r \not\models c_1 \vee \ldots \vee c_m$. We need to show that there exists a model of P that is not a model of r. By the assumption, there must be some instantiation of Y, $Y\tau$, such that $Y\tau$ satisfies $c_r\tau$ but does not satisfy any of

the $c_i\tau$'s.[6] Let $\mathcal{M}$ be the set of interpretations (for the predicates in r and P) which include $q_1\tau, \ldots, q_n\tau$ and do not include $p\tau$. Clearly, none of the interpretations in $\mathcal{M}$ are models of r. Therefore, all we need to show is that there exists an interpretation $M_0 \in \mathcal{M}$ such that M_0 is a model of P, and consequently $M(P) \not\subseteq M(r)$.

It is enough to show that $p\tau$ is not generated by P and $q_1\tau, \ldots, q_n\tau$. M_0 will then be the least model of P that consists of $q_1\tau, \ldots, q_n\tau$. To derive a contradiction, suppose that $p\tau \in P(q_1\tau, \ldots, q_n\tau)$, and let d be a derivation of $p\tau$. We can assume that d is minimal, i.e., it does not contain two identical nodes n_1 and n_2 such that n_1 is an ancestor of n_2. We create a derivation d' of pairs corresponding to d by supplementing each goal-node of d with a constraint. The constraint attached to each leaf in d is $c_r\tau$, and the constraint attached to each non-leaf node is the conjunction of the constraints of its subgoals and the interpreted literals of the rule applied. Denote the resulting derivation by d'. Clearly, all the constraints in pairs of d' are satisfiable, because $Y\tau$ satisfies c_r and is a valid derivation of $p\tau$. Furthermore, we show bottom-up induction on d', that the nodes derived in d' would be derived by our algorithm. Specifically, we show that if $(q\tau, c\tau)$ is a node in d', then the pair $(q\theta, c\theta)$ would have been derived by our algorithm, where θ is the mapping we used for the variables of r. The claim holds for the leaves of d', since they all have atoms from $q_1\tau, \ldots, q_n\tau$, and our algorithm began with the pairs $(q_1\theta, c_r\theta), \ldots, (q_n\theta, c_r\theta)$. The inductive case follows from the observation that since all argument positions have distinct variables in rules of P, any rule application that was done in d would have been done by our algorithm (because the unifications did not rely on additional equality between constants, that may have existed in $Y\tau$ and not in $Y\theta$. All equalities were made explicit as separate subgoals in the rules). However, the fact that the root of d' was derived leads to a contradiction. Since $Y\tau$ satisfies the root of d', there would be an i such that $c_i \models Y\tau$. ∎

Our bottom-up evaluation of a program with a database containing facts that are pairs of an atom and a constraint is reminiscent of the procedure used by Kanellakis et al. [Kanellakis *et al.*, 1990]. In their procedure, an EDB fact may be a generalized tuple specified in the form of a constraint on the arguments of its predicate. However, there is a key difference between the two methods. In [Kanellakis *et al.*, 1990], the constraint specifying a tuple considers *only* the arguments of the predicate involved. In our procedure, the constraint appearing in a pair is a constraint on all the constants that appear in the initial database, i.e., all the constants in $Y\theta$, where Y is the set of variables of rule r. Thus, the constraint of a pair may have constants that do not appear in the atom of that pair. The following example illustrates why their method cannot be applied for detecting uniform containment.

[6] Note that we are assuming here that the subgoals are rectified, i.e., all equality constraints on the variables in r appear in c_r.

Example 5.24: Consider rules r and s:

$r : q_1(X, Y) \wedge q_2(U, V) \Rightarrow p(X, Y)$
$s : q_1(X, Y) \wedge q_2(U, V) \wedge (U \leq V) \Rightarrow p(X, Y)$

To check whether $M(s) \subseteq M(r)$, we evaluate s with the pairs $(q_1(x_0, y_0), True)$ and $(q_2(u_0, v_0), True)$. If we use the procedure of [Kanellakis *et al.*, 1990], the result is the pair $(p(x_0, y_0), True)$, which has no recording of the fact that its derivation required that $u_0 \leq v_0$. Consequently, we will conclude erroneously that $M(s) \subseteq M(r)$ holds. In contrast, when our procedure applies rule s to the pairs $(q_1(x_0, y_0), True)$ and $(q_2(u_0, v_0), True)$, the result is the pair $(p(x_0, y_0), u_0 \leq v_0)$, making it clear that s does not contain r, because $True \not\models u_0 \leq v_0$. ∎

Complexity

The complexity of the uniform containment algorithm is in the worst case exponential in the arity of the predicates in the programs. It depends on two factors:

1. The number of pairs generated during the evaluation of $\mathcal{P}$.
2. The complexity of checking whether $c_r \models c_1 \vee \ldots \vee c_m$ holds.

The number of pairs generated during the evaluation of $\mathcal{P}$ may, in the worst case, be exponential in the number of variables of r. This is because the number of non-equivalent constraints on n constants is exponential in n. The complexity of the second part is also at most exponential in the sum of the number of variables in r and the number of constants appearing in $\mathcal{P}$.

5.3.2 Uniform Equivalence with Stratified Negation

In this section, we describe how to test uniform equivalence of datalog programs with safe, stratified negation. We begin with the case of stratified programs with neither constants nor interpreted predicates. By definition, two programs P_1 and P_2 are uniformly equivalent, denoted $P_1 \equiv^u P_2$, if for every database D (that may have both EDB and IDB facts), $P_1(D) = P_2(D)$. Note that applying a stratified program to a database that may also have IDB facts is done stratum by stratum, as in the usual case; in other words, $P(D)$ is the perfect model of the program P and the database D (cf. [Ullman, 1989]).

Suppose that P_1 and P_2 are not uniformly equivalent. Hence, there is a database D_0 such that $P_1(D_0) \neq P_2(D_0)$; D_0 is called a *counterexample*. We may assume that $P_1(D_0) \not\subseteq P_2(D_0)$ (the case $P_2(D_0) \not\subseteq P_1(D_0)$ is handled similarly).

We assume that both P_1 and P_2 have the same set of EDB predicates and the same set of IDB predicates, and moreover, there is a partition of the predicates into strata

that is a stratification for both P_1 and P_2. In particular, we assume that the lowest stratum consists of just the EDB predicates, and we refer to it as the zeroth stratum. We denote by P_1^i the program consisting of those rules of P_1 with head predicates that belong to the first i strata; similarly for P_2^i. Note that P_1^0 is an empty program (i.e., it has no rules). By definition, $P_1^0(D) = D$ for every database D; similarly for P_2^0.

We now assume that for some given i, $P_1^i \equiv^u P_2^i$, and we will show how to test whether $P_1^{i+1} \equiv^u P_2^{i+1}$. The algorithm is based on the following two lemmas.

Lemma 5.25: *Suppose that there is an i such that $P_1^i \equiv^u P_2^i$. If there is a counterexample database D_0 such that $P_1^{i+1}(D_0) \not\subseteq P_2^{i+1}(D_0)$, then there is some rule r of P_1^{i+1} with a head predicate from stratum $i+1$ and a database D, such that*

1. *D is a model of P_2^{i+1} but not a model of r; and*
2. *The number of distinct constants in D is no more than the number of distinct variables in r.*

Proof: Let $D' = P_2^i(D_0)$; note that since D' is a perfect model, $D' = P_2^i(D')$. By the assumption in the lemma, $P_1^i(D_0) = P_2^i(D_0)$ and, hence, D' is also a counterexample, i.e., $P_1^{i+1}(D') \not\subseteq P_2^{i+1}(D')$. Now let $\bar{D} = P_2^{i+1}(D')$. Observe that $\bar{D}$ and D' have the same set of facts for predicates of the first i strata, since $D' = P_2^i(D')$. In addition, observe that $D' \subseteq \bar{D}$. These observations imply that $P_1^{i+1}(D') \subseteq P_1^{i+1}(\bar{D})$. Thus, $P_1^{i+1}(\bar{D}) \not\subseteq P_2^{i+1}(\bar{D})$, because $P_1^{i+1}(D') \not\subseteq P_2^{i+1}(D')$ and $P_2^{i+1}(D') = P_2^{i+1}(\bar{D})$.

So, we have shown that $P_1^{i+1}(\bar{D}) \not\subseteq P_2^{i+1}(\bar{D})$, and $\bar{D}$ is a model of P_2^{i+1}. Therefore, there is a rule r in P_1^{i+1} of the form

$$q_1 \wedge \ldots \wedge q_m \wedge \neg s_1 \wedge \ldots \wedge \neg s_l \Rightarrow h$$

and a substitution θ such that

- the predicate of h is from stratum $i+1$,
- θ is a mapping from the variables of r to constants,
- $q_i\theta \in \bar{D}$ $(1 \leq i \leq m)$,
- $s_j\theta \notin \bar{D}$ $(1 \leq j \leq l)$, and
- $h\theta \notin \bar{D}$.

The above and the fact $\bar{D} = P_2^{i+1}(\bar{D})$ imply that the database $\bar{D}$ is a model of P_2^{i+1} but not of r^{i+1}.

Let D be the database consisting of facts from $\bar{D}$ that have only constants from $r\theta$. Database D is also a model of P_2^{i+1}. In proof, suppose that D is not a model of P_2^{i+1}. Thus, there is a rule $\bar{r}$ of P_2^{i+1} and a substitution $\bar{\tau}$, such that

1. the head $\bar{h}$ of $\bar{r}$ satisfies $\bar{h}\tau \notin D$,
2. every positive subgoal $\bar{q}$ of $\bar{r}$ satisfies $\bar{q}\tau \in D$, and
3. every negative subgoal $\bar{s}$ of $\bar{r}$ satisfies $\bar{s}\tau \notin D$.

By the definition of D, if g is a ground fact having only constants from D, then $g \in D$ if and only if $g \in \bar{D}$; moreover, for every negative subgoal $\bar{s}$, the constants appearing in $\bar{s}\tau$ are all from D, since rules are safe (cf. [Ullman, 1989]). Therefore, items (1)-(3) hold even if we replace D with $\bar{D}$, and so it follows that $\bar{D}$ is not a model of $\bar{r}$—a contradiction, since $\bar{D}$ is a model of P_2^{i+1}, and $\bar{r}$ is a rule of P_2^{i+1}. Thus, we have shown that D is a model of P_2^{i+1}. Furthermore, items (1)-(3) above imply that D is not a model of r. So, the lemma is proved. ∎

Lemma 5.26: *Suppose that $P_1^i \equiv^u P_2^i$. Moreover, suppose that there is a database D that is a model of P_2^{i+1} and is not a model of some rule r of P_1^{i+1} having a head predicate from stratum $i+1$. Then $P_1(D) \not\subseteq P_2(D)$, and, hence, $P_1 \not\equiv^u P_2$.*

Proof: From the assumptions in the lemma, it follows that rule r can be applied to D to generate a new fact g that is not already in D. Note that $g \notin P_2(D)$, since $P_2^{i+1}(D) = D$ and strata higher than $i+1$ cannot derive new facts with the same predicate as that of g. If we show that rule r can still generate g even when P_1 is applied to D, it would follow that $g \in P_1(D)$, and hence, $P_1(D) \not\subseteq P_2(D)$. To show that, recall that $P_1^i \equiv^u P_2^i$ and D is a model of P_2^{i+1}; therefore, D is also a model of P_1^i. Thus, rule r can still generate g during the application of P_1 to D, since nothing is generated by rules of lower strata. ∎

The algorithm of Figure 5.1 tests whether $P_1 \equiv^u P_2$; its correctness follows from the above two lemmas and the following proposition.

Proposition 5.27: *$P_1(D) \not\equiv P_2(D)$ if and only if there is some i and a database D such that either $P_1^i(D) \not\subseteq P_2^i(D)$ or $P_2^i(D) \not\subseteq P_1^i(D)$.*

Proof: Clearly, if $P_1(D) \not\equiv P_2(D)$ then there exists some strata i such that either $P_1^i(D) \not\subseteq P_2^i(D)$ or $P_2^i(D) \not\subseteq P_1^i(D)$. Conversely, if $P_1^i(D) \not\subseteq P_2^i(D)$, then P_1^i and P_2^i differ in some of the facts they generate from D for the first i strata. Therefore, by Lemma 5.26, $P_1(D) \not\equiv P_2(D)$. ∎

Note that in the algorithm, it does not matter what the are constants in S as long as their number is equal to the number of distinct variables in the given rule r. Also, if two databases over constants from S are isomorphic, it is sufficient to consider just one of them. In the algorithm we need to check whether a database $\bar{D}$ is a model of $\bar{P}_2$ and not of $\bar{r}$. This is done by verifying that $P_2(D) = D$ and $\bar{r}(D) \neq D$.

```
procedure check(P1, P2);
begin
    for every rule r of P1 do
    begin
        Let S be a set of v distinct constants, where v is the number of variables in r;
        for every database D that includes only constants from S do
            if D is a model of P2 but not of r then return false;
    end;
    return true;
end;
begin /* main procedure */
    for i := 1 to max-stratum do
        if not check(P1^i, P2^i) or not check(P2^i, P1^i) then return P1 ≢^u P2;
    return P1 ≡^u P2;
end
```

Figure 5.1: An algorithm for testing $P_1 \equiv^u P_2$.

Example 5.28: Let P_1 consist of the rules:

$r_1 : own(X, Y) \Rightarrow Iown(X, Y)$
$r_2 : lives(X, Z) \wedge inHouse(Z, Y) \Rightarrow Iown(X, Y)$
$r_3 : own(X, Z) \wedge lives(Y, Z) \wedge Iown(Y, U) \Rightarrow Iown(X, U)$
$r_4 : likes(X, Y) \wedge \neg Iown(X, Y) \Rightarrow buys(X, Y)$

Let P_2 consist of the rules r_1, r_4 and the rule:

$r_5 : own(X, Z) \wedge inHouse(Z, Y) \Rightarrow Iown(X, Y)$

The EDB relation *own* describes an ownership relationship between persons and objects. The IDB relation *Iown* represents a landlord's perspective of the ownership relation. The programs P_1 and P_2 are not uniformly equivalent. Specifically, consider the database D_0:

$$\{likes(a, o),\ lives(b, h),\ own(b, o),\ own(a, h)\}$$

The programs P_1 and P_2 differ already in the first stratum (in which the relation *Iown* is computed), since $Iown(a, o) \notin P_2(D_0)$ whereas $Iown(a, o) \in P_1(D_0)$. In the second stratum, when we compute the relation *buys*, we get $buys(a, o) \in P_2(D_0)$ and $buys(a, o) \notin P_1(D_0)$. ∎

To extend the algorithm to programs with interpreted predicates (and constants in the rules), we need to be careful about checking whether a database D is a model of P_2 and not of r. In order to check this we need to specify the interpretations of the interpreted predicates on the constants in S. For example, suppose r is the rule

$$e(X, Y) \wedge (X < Y) \Rightarrow p(X, Y)$$

and P_2 consists of the single rule

$$e(X, Y) \wedge (X \geq Y) \Rightarrow p(X, Y).$$

Consider the database D consisting of the fact $e(x_0, y_0)$. If $x_0 < y_0$ holds, then D is a model of P_2 and is not a model of r. However, if $x_0 > y_0$, then D is not a counterexample.

One conceptually simple (albeit not the most efficient) way to address this subtlety is to try all possible interpretations to find one in which the database is a counterexample. The interpretations can be viewed as supplements to the given database. In the case of dense-order constraints, we would do the following. Let C be the set of constants appearing in either P_1 or P_2. Instead of considering every database over constants from S, we should consider every database over constants from $S \cup C$. Moreover, for each database, we should consider every total order on the constants of the database such that the order is consistent with any order that may implicitly be defined on C (e.g., if C is a set of integers, then presumably the usual order on integers should apply to C). For each such pair consisting of a database and total order defined on its constants, we check whether the pair is a model of P_2 and not of r. Consequently, we get the following theorem:

Theorem 5.29: *Uniform equivalence for datalog programs with safe, stratified negation and interpreted predicates is decidable.*

Proof: First, it should be noted that Lemmas 5.25 and 5.26 and Proposition 5.27 hold also when the rules have interpreted literals. The only difference is that the number of objects in the counterexample may the size of $S \cup C$. All we need to show is that trying all the consistent possible interpretations of the interpreted predicates for the constants in $S \cup C$ will suffice to find a counterexample if there is one. In proof, suppose we found an interpretation I such that $D \cup I$ is a counterexample. In that case, simply replace the constants in S with constants from the domain of the interpreted predicates such that the constants satisfy the same interpreted relations as the objects in S and are consistent with I. The result will be a counterexample database. Conversely, suppose there is some counterexample database D. Simply map the constants in S to the corresponding constants in D, and interpret the interpreted predicates on S in the same way that the corresponding constants in D are interpreted.

Clearly, our algorithm would have tried that interpretation for the constants in S, and would have found the counterexample. ∎

A more efficient method for checking whether a database is a counterexample is to use an algorithm similar to the one used for uniform containment with interpreted predicates. We evaluate r and P_2 with pairs consisting of an atom and a constraint. The initial pairs are $(g, True)$, where $g \in D$. Let $(g_1, c_1), \ldots, (g_l, c_l)$ be the pairs derived by r and for which $g_i \notin D$. Similarly, let $(h_1, d_1), \ldots, (h_k, d_k)$ be the pairs derived by P_2 and for which $h_i \notin D$. Clearly, D is a model of P_2 if and only if interpretations of the interpreted predicates satisfy $\neg(d_1 \vee \ldots \vee d_k)$, since under these constraints, P_2 does not derive any new facts when applied to D. Similarly, the database D is not a model of r if the interpretations satisfy $(c_1 \vee \ldots \vee c_l)$. Therefore, D is a counterexample database for the containment of r in P_2 if and only if the constraint

$$\neg(d_1 \vee \ldots \vee d_k) \wedge (c_1 \vee \ldots \vee c_l)$$

is satisfiable.

5.3.3 Beyond Uniform Containment

For testing uniform containment of $\mathcal{P}_1$ in $\mathcal{P}_2$, it is enough to check the containment separately for each rule of $\mathcal{P}_1$. Consequently, uniform containment completely ignores possible interactions between the rules which may imply containment of $\mathcal{P}_1$ in $\mathcal{P}_2$. Consider the following example.

Example 5.30: Consider the following programs whose query predicate is p. Let $\mathcal{P}_1$ be:

$r_1 : q(X) \wedge (X < 5) \Rightarrow p(X)$
$r_2 : e(X) \wedge (X > 0) \Rightarrow q(X)$

And let $\mathcal{P}_2$ be the program:

$r_3 : q(X) \wedge (X < 6) \wedge (X > 0) \Rightarrow p(X)$
$r_4 : e(X) \wedge (X > 0) \Rightarrow q(X)$

The program $\mathcal{P}_1$ is contained in $\mathcal{P}_2$, because whenever $0 < X < 5$, the atom $p(X)$ will be derived from $\mathcal{P}_2$ if $e(X)$ is in the database. However, r_1 is not uniformly contained in $\mathcal{P}_2$ (and, therefore, $\mathcal{P}_1 \not\subseteq^u \mathcal{P}_2$). For example, the model consisting of $\{q(-1), e(-1), \neg p(-1)\}$ is a model of $\mathcal{P}_2$ but not a model of $\mathcal{P}_1$. ∎

The weakness of the uniform containment test stems from the fact that it considers containment of the set of all models, while in order to prove (ordinary) containment,

it is sufficient to consider containment of only the minimal models.[7] Therefore, there may be cases in which containment of minimal models holds, but containment of all models does not. To get a stronger test, we may try to transform $\mathcal{P}_1$ into an equivalent program $\mathcal{P}'$ with a larger set of models (but, of course, the same set of minimal models, since equivalence must be preserved). We may then be able to show that $M(\mathcal{P}_2) \subseteq M(\mathcal{P}')$ holds, where $M(\mathcal{P}_2) \subseteq M(\mathcal{P}_1)$ failed. One way of doing this transformation is by propagating constraints from one rule to another by using the rules given by the query-tree. In our example, the result of constraint propagation is the following program $\mathcal{P}'$:

r_1' : $q(X) \wedge (X < 5) \wedge (X > 0) \Rightarrow p(X)$
r_2' : $e(X) \wedge (X > 0) \Rightarrow q(X)$

Now we can show that $\mathcal{P}' \subseteq^u \mathcal{P}_2$, and since $\mathcal{P}_1 \equiv \mathcal{P}'$, it follows that $\mathcal{P}_1 \subseteq^u \mathcal{P}_2$.

5.4 Conclusions

In this chapter we studied the problem of detecting independence of queries from updates. We provided insight into the problem by relating it to the problems of detecting irrelevance and equivalence of datalog programs. As a consequence of this connection, we made several contributions:

1. Provided algorithms that guarantee sufficient conditions for detecting independence, based on strong irrelevance.

2. Showed additional cases in which detecting independence is decidable, and gave efficient algorithms for doing so.

3. Showed cases where independence of an insertion is equivalent to independence of a deletion, thereby making the latter easier to compute.

Viewing the problem of independence from the perspective of irrelevance and equivalence also suggests that further algorithms for independence can be found by considering other sufficient conditions for weak irrelevance and equivalence. One such direction, based on extending uniform equivalence, was discussed in Section 5.3.3. Other sufficient conditions can be achieved by considering strong irrelevance based on minimal derivations, as described in Chapter 3. This chapter also made contributions to the problem of detecting equivalence of datalog programs, which is a recurring problem in query optimization.

[7] In our formalism, a set of relations for the EDB and IDB predicates is a minimal model if the IDB part is a minimal model once the EDB facts are added to the program as rules with empty bodies.

5.4.1 Related Work

As discussed throughout the chapter, Blakeley et al. [Blakeley *et al.*, 1989] and Elkan [Elkan, 1990] have studied the problem of independence. In summary, they have considered the problem for restricted languages in which strong irrelevance is the same as weak irrelevance. Blakeley et al. consider non-recursive knowledge bases without interpreted predicates. Elkan generalizes these results and provides a decision procedure for strong irrelevance in the case of non-recursive knowledge bases. Our results extend Elkan's by providing a decision procedure for strong irrelevance (and therefore, independence) for arbitrary datalog programs. Furthermore, we show additional cases in which independence is decidable, specifically for arbitrary non-recursive knowledge bases. It should be noted that Elkan also suggested a proof method for detecting independence in the recursive case; however, he provides no characterization of the power of that proof method, but it should be noted that it cannot capture all cases detected by the query-tree.

Our work also generalizes previous work on containment of conjunctive queries with interpreted predicates by Klug [Klug, 1988]. Klug showed that if all the constraints are *left-semiinterval* or all constraints are *right-semiinterval*,[8] then containment of conjunctive queries can be decided by finding a homomorphism from one query to the other. For general conjunctive queries, he pointed out that it could be done by finding a homomorphism for every possible ordering of the variables and constants in the queries. The number of such orderings is exponential in the number of variables appearing in the constraints. Recently, van der Meyden [van der Meyden, 1992] has shown that the containment problem of conjunctive queries with order constraints is Π_2^p-complete. In our algorithm, the complexity depends only on the number of orderings that are actually generated during the evaluation of $\mathcal{P}$. More precisely, our algorithm generates partial rather than complete orderings of the variables and constants in the queries. Essentially, it lumps together complete orderings that need not be distinguished from each other in order to test containment. Therefore, our algorithm is likely to be better in practice, albeit not in the worst case. Of course, our algorithm also applies to more than just conjunctive queries by considering recursive programs as well.

[8] See [Klug, 1988] for precise definitions of these restrictions.

Chapter 6

Irrelevance and Abstractions

6.1 Introduction

In the previous chapters we discussed irrelevance claims whose subject was formulas. When we detected that a formula was irrelevant to a query, that served as a justification for ignoring the formula when we searched for for an answer to the query. Ignoring an irrelevant formula can be viewed as a simple way of abstracting a knowledge base. This view suggests that more interesting abstractions can be obtained by considering other kinds of irrelevance claims, based on different subjects of irrelevance.

Research on reasoning with abstractions focuses on finding abstractions that will yield more efficient inference. The intuition underlying much of that research is that a *good* abstraction is one that removes irrelevant detail. If the details removed in the abstraction are indeed irrelevant, then the solutions obtained from the abstract knowledge base will hold in the original knowledge base, or can be refined in a structured fashion to solutions from the original knowledge base.

This chapter proposes an approach to research on abstractions that exploits the connection between the notion of irrelevance and the creation of abstractions. It makes the first steps in formalizing this connection and describes the possible payoffs from the approach. We begin by illustrating two new irrelevance subjects and the abstractions that they justify. The potential payoff of the proposed approach is demonstrated by considering these examples in detail in Sections 6.3 and 6.4. The first example illustrates the irrelevance of *predicate arguments*.

Example 6.1: Consider the following rules describing flight routes between cities. The third argument of *flight* and *route* denote costs of the flights, and their fourth arguments denote the airline of the flight/route.

$r_1 : flight(X, Y, C, A) \Rightarrow route(X, Y, C, A)$
$r_2 : flight(X, Z, C_1, A) \wedge route(Z, Y, C_2, A) \Rightarrow route(X, Y, C_1 + C_2, A)$

$r_3 : route(X, Y, C, A) \Rightarrow airlineFlight(X, Y, A)$

The knowledge base also contains a set of ground atoms for the predicate *flight*. The atom $airlineFlight(X, Y, A)$ denotes that there is a route (i.e., some sequence of flights) from X to Y that uses only airline A. Suppose we want to query the knowledge base for the existence of a flight from SF to LA on K_2 airlines, i.e., $airlineFlight(SF, LA, K_2)$. The costs of flights are irrelevant to this query, i.e., the third argument of *flight* and *route* can be projected out to yield a smaller knowledge base and search space. Specifically, we can rewrite the rules as follows:

$r'_1 : flight'(X, Y, A) \Rightarrow route'(X, Y, A)$
$r'_2 : flight'(X, Z, A) \wedge route'(Z, Y, A) \Rightarrow route'(X, Y, A)$
$r'_3 : route'(X, Y, A) \Rightarrow airlineFlight(X, Y, A)$

We also project out the third argument in the ground atoms of the predicate *flight*. Consequently, multiple flight facts describing different fares for the same flight are collapsed to one fact in the knowledge base. As a result, the knowledge base will contain fewer facts and simpler rules and therefore the space searched may be significantly smaller than in the original one. For example, consider the difference between the rules r_2 and r'_2. In rule r_2, if we fail to join a ground atom $flight(x, z, c_1, a)$ with a ground atom $route(z, y, c_2, a)$, the backward chainer might still try to join the atom $flight(x, z, c'_1, a)$ with an appropriate atom of *route* for every value c'_1 for which it finds an atom in the flight database, and will fail on all of them. In contrast, rule r'_2 will not try other costs for the same flight route.

Although in some simple cases these repetitions can be eliminated by employing some method of dependency directed backtracking, such methods will not be as general as projecting out arguments and will also have additional costs associated with maintaining the dependencies. ∎

The following example illustrates the irrelevance of a *predicate refinement*:

Example 6.2: Consider a knowledge base with the following formulas:

$r_1 : sportsCar(X) \Rightarrow car(X)$
$r_2 : familyCar(X) \Rightarrow car(X)$
$r_3 : car(X) \Rightarrow vehicle(X)$
$r_4 : bicycle(X) \Rightarrow vehicle(X)$
$r_5 : sportsCar(X) \Rightarrow highRiskInsurance(X)$
$r_6 : car(X) \Rightarrow hasMotor(X)$
$r_7 : vehicle(X) \wedge hasMotor(X) \Rightarrow motorizedVehicle(X)$
$g_1 : familyCar(Camry)$.

Consider the query $motorizedVehicle(Camry)$. With respect to that query, the refinement between sports cars and family cars is irrelevant. Intuitively, all that matters about the Camry is that it is a car. Consequently, the query can be solved if we abstract the representation of the domain by removing the refinement between the predicates $sportsCar$ and $familyCar$. To do so, we remove rules r_1 and r_2 and replace g_1 by the formula:

$g_1' : car(Camry)$

The rule r_5 is also removed because it *distinguishes* between the different types of cars, and is therefore irrelevant. Abstracting the representation will yield more efficient inference for several reasons. First, it is no longer necessary to derive that a Camry is a car. In the example, there are only two rules that may be used to derive that, but in general, there may be many different subclasses of cars and the cost of deriving $car(Camry)$ may be arbitrarily large. Second, by removing the formulas that distinguish between the types of cars, we reduce the size of the space that needs to be searched. ∎

Recently, research on abstractions and approximations has received renewed attention [Ellman, 1990; Ellman, 1992; Lowry, 1992]. However, two key problems in this field remain largely open. The first is how a system can automatically create an abstraction that is well suited to a particular query. The second challenge is understanding the utility of reasoning with multiple levels of abstraction. Our approach addresses these issues as follows. When an abstraction is being considered, our approach is to articulate which knowledge is being removed in the process of the abstraction and to justify the abstraction by the fact that this knowledge is irrelevant to the query at hand. Reasoning about abstractions then becomes a problem of reasoning about irrelevance. The formal analysis of irrelevance will give us several insights into the corresponding abstractions:

1. The problem of automatically generating abstractions becomes well defined as a problem of automatically deriving irrelevance claims. Often this can be done by using existing algorithms for automatically deriving irrelevance claims, as we see in Sections 6.3 and 6.4.

2. Understanding the utility of exploiting irrelevance claims gives us insight into the utility of the abstraction based on it. For example, if an abstraction is based on a weak irrelevance claim, then it is not necessarily computationally advantageous, whereas if it is based on a strong irrelevance claim, it is guaranteed to lead to computational savings. Furthermore, the underlying irrelevance claims can indicate whether abstractions can be composed, based on composing the underlying irrelevance claims.

3. The ability to explicitly state the irrelevance claims underlying abstractions provides us with a formalism in which we can reason *about* abstractions. This is useful in several scenarios:

   - In general, we cannot expect a single abstraction hierarchy to be well suited for all possible queries. Therefore we need to tailor our abstractions for specific queries, and in doing so we can often be aided by additional knowledge about the domain. Expressing such knowledge in the form of irrelevance claims and combining it with other knowledge about the domain provides a powerful mechanism for incorporating domain knowledge into the creation of abstractions.
   - Several problem solving scenarios give rise to situations in which we are given multiple descriptions of aspects of a domain at varying levels of abstraction. In such a situation our task is twofold. First, we need to select the level of abstraction that is best suited for a given query, and second, we need to combine descriptions of different aspects of the domain to create one coherent and consistent description. By stating explicitly the assumptions underlying the multiple descriptions, we can reason about their consistency and adequacy. The following are examples of such scenarios:
     (a) **Reasoning about physical systems:** In this domain (discussed in detail in Chapter 7) we are given descriptions of physical phenomena in the world at different levels of abstraction. Our task is to compose descriptions of relevant aspects of the system such that we can answer a query about a given system. For example, suppose we are composing a representation for a given device that includes a battery connected to a wire, each of which can be described at different levels of abstraction. In particular, we can describe each of them under the assumption that their electrical properties are irrelevant or without that assumption. Reasoning about the assumptions underlying these descriptions will ensure that we do not compose a description of the battery that ignores its electrical properties (e.g., its voltage) with a description of the wire that considers the voltage of the battery relevant.
     (b) **Reasoning with contexts:** Contexts [Guha, 1991] are small theories that describe limited aspects of the world. A knowledge base describing a complex domain can benefit from being divided into contexts both in simplicity of representation and efficiency of reasoning. Here too, answering a query requires that we decide which contexts are relevant to the query and are consistent with each other. This can be done by reasoning with explicit statements about the assumptions underlying each context.

(c) **Distributed heterogeneous databases:** A distributed database (whether centrally managed or employing a federated architecture) may contain several databases with overlapping data. The databases may describe the data with different levels of abstraction and assumptions. As in the previous two cases, given a query, the system must find the (parts of the) databases that are needed to answer the query and must combine knowledge from the different databases to provide a coherent answer. An additional challenge here is to minimize the costs that may be associated with accessing remote databases.

It should be noted that in the latter two examples, before we can reason about the abstractions underlying the different contexts or databases, we must resolve the semantic mismatches between the descriptions used in each context/database.[1]

6.2 New Irrelevance Subjects

As explained, abstractions can be obtained by considering new kinds of irrelevance subjects. This section describes informally several such subjects and shows how they account for abstractions with which we are familiar. The irrelevance subjects that we discuss are broadly divided into two classes, one that concerns relations in the domain (and their corresponding predicate symbols) and one involving objects in the domain. The irrelevance subjects concerning relations include the following:

- **Predicate irrelevance:** We may state that a certain predicate (representing a relation in the domain) is irrelevant to a query. For example, if we are modeling the behavior of a battery for a short period of time, the property of being a rechargeable battery is irrelevant. Such an irrelevance claim can justify simplifying formulas by removing literals containing the irrelevant predicate (or by removing formulas completely).

- **Predicate Refinement:** Irrelevance of predicate refinements can appear in two forms. In the first, illustrated in Example 6.2 (and further investigated in Section 6.4), we have a set of predicates $q_1, \ldots, q_n$ and an irrelevance claim stating that the given q_i is irrelevant. That means we only need to know that a certain object (or tuple of objects) belongs to one of the q_i's, but not which one. Consequently, the abstraction will replace $q_1, \ldots, q_n$ by a new predicate q intended to denote the union of the interpretations of $q_1, \ldots, q_n$.

[1] In research on modeling physical systems, most works make the assumption that all the descriptions of the domain are based on one consistent ontology.

In the second form, we want to remove $q_1, \ldots, q_n$ because we are only interested in objects that belong to *all* the q_i's. Consequently, we will replace $q_1, \ldots, q_n$ by a predicate that denotes the intersection of the relations denoted by $q_1, \ldots, q_n$. For example, we may have two predicates *adult* and *citizen*. However, in a theory that represents the domain of elections, we wish to represent only objects that satisfy the intersection of these two relations. Such an abstraction will enable us to remove many formulas as well as save computation of intersections (or more generally, joins of relations).

- **Predicate argument:** As we saw in Example 6.1, we can often simplify a representation by reducing the arity of some predicates (i.e., projecting them on a subset of their arguments). Section 6.3 will discuss this subject in detail.

Irrelevance subjects concerning objects in the domain include the following:

- **Object irrelevance:** We may state that a certain object in the domain is irrelevant to a query. For example, we may state that the battery of the car is irrelevant to a query regarding its transmission system. Consequently, we can ignore formulas in the knowledge base, that include constants (or terms) denoting the irrelevant object.

- **Object refinement:** As with predicate refinements, we can state that a refinement between objects is irrelevant to a query. Given a set of objects $a_1, \ldots, a_n$, we can replace them with a single object a. As in the case of predicate refinements, there are two ways we can interpret a. The first is to assume that a has only those properties that are common to each of the a_i's, and the second is to assume that a has any property that any of the a_i's has.[2] For example, suppose we are reasoning about a chemical reaction. We do not need to represent each molecule in a given solution. Instead, we reason with one representative molecule of each different type. We ascribe to a representative molecule all the properties that are shared by all molecules of its type.[3]

- **Object aggregation:** A common abstraction that arises in many contexts is aggregation. Instead of representing a set of objects, we represent only a single object denoting their aggregate. For example, instead of representing the parts of a chair, we can represent the chair as a single object. We can use this representation when the properties that are relevant to the query are only those that apply to the aggregate and not to its subparts. Note that object

[2] Of course, special care must be given to formulas of the form $a_1 \neq a_2$.

[3] One can also devise methods for abstracting object refinements that lie in the middle of these two extremes. For example, we can associate with the representative object only the properties that are *typical* of the set it is representing.

aggregation is different from object refinement. Here, the new object represents the aggregation of a set of objects, while in the case of object refinement the new object denotes a representative of objects in a set.

- **Object homogeneity:** In object refinement we replaced a set of objects by a single one. However, in some cases we may need to retain all the objects (e.g., their number is important), but we want to represent them as a set of homogeneous objects. This means that we abstract all the differences between them except for object identity. For example, consider the 15 puzzle. A powerful heuristic for solving the puzzle is to place the first tile in place, and then proceed to place the second, (while keeping the first in a fixed position), etc. For the subgoal of placing the first tile in place, there is no need to distinguish between the tiles 2-15. Two states that differ only in the location of one of these tiles should be indistinguishable. Making this abstraction reduces the number of possible states from 16! ($\approx 10^{13}$) to 15 x 16 = 240 states.

In addition to irrelevance subjects concerning relations and objects, we can also consider subjects that abstract function symbols. In some domains, we can also consider more specific subjects. For example, in planning we can consider irrelevance of states, actions or action preconditions

6.2.1 Defining Irrelevance of New Subjects

As a basis for the approach we are proposing, we need to make a formal connection between abstractions and irrelevance. This section shows how irrelevance of new subjects can be formalized in the framework we discussed in Chapter 2.

The definitions of irrelevance that we have considered were based on the intuition that a subject ϕ is irrelevant to a query if ϕ can be *removed* without changing the answer to the query. In the case of ϕ being a formula, removing ϕ meant literally removing it from the knowledge base (or revising the knowledge base so it does not entail ϕ). For the new irrelevance subjects, although we have an intended abstraction in mind for removing ϕ, the actual removal involves subtle details. For example, in the case of irrelevance of a predicate refinement $\{q_1, \ldots, q_n\}$, we would remove it by replacing all occurrences of q_i by a new predicate q. The intended interpretation of q is the union of the interpretations of $\{q_1, \ldots, q_n\}$. However, in doing so we must be careful. For example, if we are removing the distinction between the predicates $\{sportsCar, familyCar\}$, and we have the formula

$$familyCar(X) \Rightarrow \neg sportsCar(X)$$

then performing the substitution would result in the contradiction

$car(X) \Rightarrow \neg car(X)$.

Assuming that we have some function Abs_ϕ for abstracting a given knowledge base Δ which does not introduce unwanted formulas, our intuition of irrelevance would imply that ϕ is irrelevant to a query q w.r.t. a knowledge base Δ if

$$\Delta \vdash q \iff Abs_\phi(\Delta) \vdash Abs_\phi(q). \tag{6.1}$$

In words, ϕ is irrelevant to the query q if abstracting the representation by removing ϕ (resulting in the knowledge base $Abs_\phi(\Delta)$) does not change the derivability of the query. As we saw in Chapter 2, a more refined account of irrelevance, based on a proof theoretic analysis, enables us to address the key issues regarding irrelevance. In this case, we want our analysis to help in developing algorithms for automatically justifying and creating abstractions and in analyzing the utility of reasoning with abstractions.

We now extend the framework described in Section 2.3 to new irrelevance subjects. Recall that a definition of irrelevance in our space was obtained by considering some condition DI (which depended on the subject ϕ) over a chosen set of derivations of the query $\mathcal{D}_0$. We said that ϕ is strongly irrelevant to a query q if $DI(\phi, D)$ holds for all derivations $D \in \mathcal{D}_0$, and that it is weakly irrelevant to q if $DI(\phi, D)$ holds for some derivation $D \in \mathcal{D}_0$. To extend the framework, we consider appropriate definitions of the predicate DI.

The definitions we consider for DI[4] are based on identifying formulas that are *independent* of the irrelevance subject ϕ. Intuitively, a formula is independent of ϕ if it does not rely on ϕ, i.e., it holds even if ϕ is removed. The definition of independence will also be used to define the abstract knowledge base $Abs_\phi(\Delta)$ such that it does not introduce unwanted formulas. Specifically, $Abs_\phi(\Delta)$ will contain the abstractions of the formulas in Δ that are independent of ϕ. Formal definitions of independence will be given in subsequent sections. The following examples motivate the concept.

Consider Example 6.2 and the predicate refinement $\{sportsCar, familyCar\}$ and the rule

$r : sportsCar(X) \Rightarrow vehicle(X)$.

The rule r is independent of the predicate refinement, because if we replaced occurrences of $sportsCar$ by $familyCar$, the resulting rule

$r' : familyCar(X) \Rightarrow vehicle(X)$

also follows from the knowledge base. Therefore, replacing r by the rule

$car(X) \Rightarrow vehicle(X)$

[4] And we do not claim that these are the only viable definitions.

would not contradict our previous knowledge or add to it. In contrast, the rule

$s : sportsCar(X) \Rightarrow highRiskToInsure(X)$

is not independent of the predicate refinement because the rule:

$s' : familyCar(X) \Rightarrow highRiskToInsure(X)$

does not follow from the KB. Based on a definition of independence, we can consider several definitions of DI in the same fashion we did in Section 2.3:

Definition 6.3:

- $DI_1(\phi, D)$ if $Base(D)$[5] does not contain any formula that is not independent of ϕ.
- $DI_2(\phi, D)$ if D does not contain any formula that is not independent of ϕ.
- $DI_3(\phi, D)$ if $Base(D)$ does not entail any formula that is not independent of ϕ.

∎

Returning to Example 6.2, the predicate refinement $\phi = \{sportsCar, familyCar\}$ is strongly irrelevant to the query $q = motorizedVehicle(Camry)$ because the (single) derivation of q uses only formulas that are independent of ϕ. On the other hand, ϕ is not strongly (or weakly) irrelevant to the query $q_1 = highRiskToEnsure(X)$ because derivations of q_1 will use the rule r_5.

In the following sections, we consider specific definitions of independence (and irrelevance) and show how they are used to develop algorithms for automatically creating abstractions.

6.3 Irrelevance of Predicate Arguments

As illustrated in Example 6.1, it is sometimes possible to simplify a representation by projecting out arguments of some predicates, thereby reducing their arity and leading to more efficient reasoning. Intuitively, we can project out the arguments if they are irrelevant to a query. An argument is irrelevant to a query if the solution of the query requires *some* values for that argument, but the actual values used are not important, and they do not have to satisfy any other constraints. In this section, we will formalize the irrelevance of a predicate argument.

We denote a set of predicate arguments by $\mathcal{R}$, which is a set of pairs (q_i, n_i), where q_i is a predicate and n_i is an integer less or equal to the arity of q_i. For example, the

[5] Recall that $Base(D)$ is the set of formulas in the leaves of the proof tree (see Section 2.3)

set $\{(flight, 3), (route, 4)\}$ represents the third argument of the predicate *flight* and the fourth argument of the predicate *route*. In projecting out the set $\mathcal{R}$, we perform the following syntactic transformation $f_{\mathcal{R}}(\phi)$ to a formula ϕ in the knowledge base:

> Let $p(\bar{X}_1), \ldots, p(\bar{X}_m)$ be the literals of the predicate p in ϕ (both positive and negative). Suppose the arity of p is l and that $(p, i_1), \ldots, (p, i_k) \in \mathcal{R}$. We introduce a new predicate p' of arity $l-k$. We replace each atom $p(\bar{X}_i)$ of p by an atom $p'(\bar{Y}_i)$, where $\bar{Y}_i$ is the result of projecting arguments $i_1, \ldots, i_k$ out of $\bar{X}_i$. Note that p' may be of arity 0. If p is an order predicate we replace its atoms by *True*.

The function $f_{\mathcal{R}}$ is extended in a straightforward manner to sets of formulas. Note that if a predicate p appears in $\mathcal{R}$, then we replace it with the same new predicate in every formula. As explained earlier, simply applying the substitution $f_{\mathcal{R}}$ to all formulas in the knowledge base may introduce inconsistencies. We therefore apply the substitution only to formulas that are independent of $\mathcal{R}$. Our definitions of irrelevance are also based on the notion of independence.

Our definition of independence is based on the semantics of the abstraction we are performing. Specifically, if a predicate p denotes a relation P, and we project out some of the arguments of p, then the resulting predicate should denote the corresponding projection of P. Given an interpretation I for the symbols in a knowledge base Δ, we define an interpretation $Abs(I)$ for the symbols in $f_{\mathcal{R}}(\Delta)$ as follows:

- The interpretations of terms in I and $Abs(I)$ are identical.
- If the predicate p does not appear in $\mathcal{R}$, then p is mapped to the same relation as in I.
- If $(p, i_1), \ldots, (p, i_k) \in \mathcal{R}$, and p was mapped to the relation P, then the predicate p' is mapped to the relation P', where P' is the result of projecting the arguments $i_1, \ldots, i_k$ out of P.

Independence is defined as follows:

Definition 6.4: A formula ψ is independent of the predicate arguments $\mathcal{R}$ if for any interpretation I:

$$I \models \Delta \Longrightarrow Abs(I) \models f_{\mathcal{R}}(\psi).$$

Intuitively, a clause C is independent of $\mathcal{Q}$ if the formula $f_{\mathcal{Q}}(C)$ does not impose any additional constraints on the possible states of the world, as described by Δ (and therefore does not add any new knowledge). We define the abstract knowledge base resulting from removing $\mathcal{R}$ from Δ, denoted by $Abs_{\mathcal{R}}(\Delta)$. It includes the abstractions of formulas in Δ that are independent of $\mathcal{R}$, i.e.,

$$Abs_{\mathcal{R}}(\Delta) = \{f_{\mathcal{R}}(v) \mid v \in \Delta \text{ and } v \text{ is independent of } \mathcal{R}\}.$$

In Example 6.1, the rule

$$r_2 : flight(X, Z, C_1, A) \wedge route(Z, Y, C_2, A) \Rightarrow route(X, Y, C_1 + C_2, A)$$

is independent of the arguments $\{(flight, 3), (route, 3)\}$, but is not independent of the arguments $\{(flight, 4), (route, 4)\}$. To see the latter, consider the interpretation I in which $flight$ denotes the single tuple relation $\{(a, b, 1, twa)\}$ and $route$ denotes the relation $\{(b, c, 1, united)\}$. The abstract interpretation $Abs(I)$ will map $flight'$ to $\{(a, b, 1)\}$ and $route'$ to $\{(b, c, 1)\}$. Therefore, I is a model of r_2, but $Abs(I)$ is not a model of r_2'.

Based on independence, we can define irrelevance using the definitions of DI given in Definition 6.3. For example, we can define $\mathcal{R}$ to be weakly irrelevant to a query q if there is some derivation of q that contains only formulas independent of $\mathcal{R}$. The following theorem shows that weak irrelevance provides a logical justification for the abstraction that fits our intuitions stated in Equation 6.1, i.e., provides a justification for using $Abs_{\mathcal{R}}(\Delta)$ instead of Δ:

Theorem 6.5: *Let $\mathcal{D}_q(\Delta)$ be the set of derivations of the query q from the knowledge base Δ. If $WI(\mathcal{R}, q, \Delta, DI_1, \mathcal{D}_q)$ holds then*

$$\Delta \vdash q \Longrightarrow Abs_{\mathcal{R}}(\Delta) \vdash f_{\mathcal{R}}(q)$$

and, if q contains no irrelevant arguments (i.e., $q = f_{\mathcal{R}}(q)$) then

$$Abs_{\mathcal{R}}(\Delta) \models f_{\mathcal{R}}(q) \Longrightarrow \Delta \models q.$$

Note that if we have a complete set of inference rules (e.g., backward chaining for atomic queries in Horn rule knowledge bases), then the above theorem implies[6]

$$\Delta \vdash q \Longleftrightarrow Abs_{\mathcal{R}}(\Delta) \vdash f_{\mathcal{R}}(q).$$

[6]Note that we are assuming throughout that our inference rules are sound

Automatically Deriving Irrelevance of Predicate Arguments

The following observation follows from the definitions of irrelevance and will form the basis for algorithms for deriving irrelevance of predicate arguments:

Observation 6.6: Let Φ be a set of clauses such that $WI(\Phi, q, \Delta, DI_1, \mathcal{D}_0)$ holds, and such that all clauses in $\Delta - \Phi$ are independent of the arguments $\mathcal{R}$. Then $WI(\mathcal{R}, q, \Delta, DI_1, \mathcal{D}_0)$ holds, and moreover, we can abstract Δ by $Abs_{\mathcal{R}}(\Delta - \Phi)$.

Therefore, to derive irrelevance of a set of predicate arguments $\mathcal{R}$, our strategy will be to find a set of clauses Φ such that Φ is weakly irrelevant to q w.r.t. Δ and all the clauses in $\Delta - \Phi$ are independent of $\mathcal{R}$. Note that ground atomic formulas are always independent of any set of predicate arguments. Therefore, we need only consider clauses that are not ground atomic, and our results will be independent of any changes made to ground atomic clauses in our knowledge base.

Finding a set Φ can be done using any of the methods described so far. For example, in the case of Horn rule knowledge bases, the query-tree can be used to find all the formulas Φ that are strongly irrelevant to the query, and (since strong-irrelevance entails weak irrelevance), the formulas Φ are weakly irrelevant to the query as well. The algorithms in Chapter 5 can be used to detect additional weakly irrelevant formulas that are not detected by the query-tree. Finally, for general clause form knowledge bases, we can use connection graphs to derive sufficient conditions for strong irrelevance.

To derive irrelevance of predicate arguments, we need to check that the formulas in the set $\Delta - \Phi$ are independent of a set of predicate arguments $\mathcal{R}$. To facilitate this check, the following theorem provides a syntactic condition for independence. We assume that a formula C is given in clause form (cf. [Genesereth and Nilsson, 1987]). A literal in a clause is negative if it is a negation of an atomic formula (e.g., $\neg q(X)$ is a negative literal, while $p(X, Y)$ is a positive literal). $Neg(C)$ ($Pos(C)$) denotes the set of negative (positive) literals in a clause C. We assume that $Neg(C)$ contains only simple terms, i.e., variables or constants. $Pos(C)$ can contain arbitrary terms. Given a clause C, we denote by $AtPos(p, i, C)$ the set of terms that appear in the ith position of occurrences of p in C.

Theorem 6.7: *Let C be a clause and $\mathcal{R}$ be a set of arguments. The clause C is independent of $\mathcal{R}$ if the following condition holds for every term τ in the set $AtPos(p, i, C)$ for every $(p, i) \in \mathcal{R}$:*

A1. If τ is not a variable, then it does not appear in $Neg(c)$.

A2. If τ is a variable, then it has at most a single appearance in $Neg(C)$.

A3. If τ is a variable appearing in $Neg(C)$ and appears also in a term in position j of a predicate q in $Pos(C)$, then $(q, j) \in \mathcal{R}$.

The proof is given in Appendix A.[7] It is easy to see that if C is a clause for which the set of arguments $\mathcal{R}_1$ and $\mathcal{R}_2$ satisfy conditions A1,A2 and A3, then the set $\mathcal{R}_1 \cup \mathcal{R}_2$ will satisfy these conditions too.

Given a set of formulas Ψ and a set of arguments $\mathcal{R}$, we can simply check whether each of the formulas in Ψ is independent of $\mathcal{R}$ using the conditions of Theorem 6.7. However, a more interesting question is how we can automatically find the maximal set $\mathcal{R}$ of arguments such that each of the formulas in Ψ is independent of $\mathcal{R}$, given a specific query q.[8] We now describe an algorithm for finding such a set $\mathcal{R}$.

Given a clause C and an argument i of a predicate p that appears in C, two things may happen. The first is that there is no set of arguments $\mathcal{R}$ such that $(p, i) \in \mathcal{R}$ and C is independent of $\mathcal{R}$. In this case we will say that (p, i) is *needed* in C. Otherwise, we denote by $PC(C, p, i)$ the minimal such set of arguments. Note that $PC(C, p, i)$ is unique since it can be determined by repeatedly applying condition A3. Furthermore, note that $PC(C, p, i)$ can be the singleton set containing (p, i).

Our algorithm starts out assuming that every argument of every predicate, except for the query predicate, is irrelevant to the query. It makes one pass over the clauses in Ψ and either removes arguments from the list of irrelevant arguments, or adds preconditions for the inclusion of other arguments in the list. Finally, it removes from the irrelevant list any argument whose preconditions are not satisfied. The algorithm is shown in Figure 6.1.

Consider the application of the algorithm to the rules in Example 6.1, with the query $airlineFlight(SF, LA, K_2)$. The set $\mathcal{R}$ initially includes all the arguments of *route* and *flight*. When considering the rule r_1, the algorithm adds the argument $(route, i)$ to the preconditions of $(flight, i)$, for $i = 1, \ldots, 4$. Considering rule r_2, the algorithm removes the arguments $(flight, 2), (flight, 4), (route, 1)$ and $(route, 4)$ from $\mathcal{R}$. As a consequence, the argument $(flight, 1)$ is removed from $\mathcal{R}$ because its precondition was removed. Finally, in considering rule r_3, the argument $(route, 2)$ is removed from $\mathcal{R}$ because the argument $(airlineFlight, 2)$ is not a member of $\mathcal{R}$. Therefore, the algorithm returns that the arguments $(flight, 3)$ and $(route, 3)$ are irrelevant to the given query.

---

[7] Note that the conditions A1, A2 and A3 are not necessary conditions for independence. However, a necessary condition is considerably more elaborate and is not presented here. For example, the rule

$$p(X, X, Y) \Rightarrow q(X, X, Y)$$

is independent of the set of arguments $\{(p, 1), (p, 2), (q, 1), (q, 2)\}$, but condition A2 is not satisfied

[8] Note that the arguments of q are assumed to be relevant to the query

procedure find-irrelevant-arguments(Ψ, q)
begin /* Ψ are the clauses and q is the query predicate */
$\mathcal{P}$ = The predicates appearing in Ψ, and not in q.
$\mathcal{R} = \{(p, i) \mid p \in \mathcal{P}$ and i is an argument of $p\}$.
for every $s \in \mathcal{R}$, Preconditions(s)= {}.
for every $C \in \Psi$ **do:**.
for every $(p, i) \in \mathcal{R}$
if p appears in C and (p, i) is needed in C **then**
remove (p, i) from $\mathcal{R}$.
else
if $PC(C, p, i) \not\subseteq \mathcal{R}$ **then** remove (p, i) from $\mathcal{R}$.
else Preconditions $((p, i))$ = Preconditions $((p, i)) \cup PC(C, p, i) - \{(p, i)\}$.
repeat
if $(p, i) \in \mathcal{R}$ and $(q, j) \in$ Preconditions $((p, i))$ and $(q, j) \notin \mathcal{R}$ **then**
remove (p, i) from $\mathcal{R}$.
until no changes are made to $\mathcal{R}$.
return $\mathcal{R}$.
end.

Figure 6.1: Algorithm for finding irrelevant predicate arguments

The algorithm finds the maximal set of predicate arguments that satisfies conditions A1, A2 and A3. This follows from the observation that for every argument in the returned set, its precondition arguments are also in $\mathcal{R}$. Furthermore, every argument that was removed from $\mathcal{R}$ was either needed in some clause or required some other argument that is not a member of $\mathcal{R}$.

To summarize this section, we have presented formal definitions of irrelevance of predicate arguments. As a result, we were able to develop an algorithm for automatically deriving such irrelevance claims. The formalization also gives us insight into the utility of removing predicate arguments. Finally, we can also devise algorithms for deriving logical conclusions from external irrelevance claims. If we are told that the arguments $\mathcal{R}$ are irrelevant to a query q, we remove from the knowledge base all the formulas that are not independent of $\mathcal{R}$. We then apply our algorithms to the remaining set of formulas to derive additional irrelevance claims.

6.4 Irrelevance of Predicate Refinements

A predicate refinement is a set of predicates of equal arity $\mathcal{Q} = \{q_1, \ldots, q_n\}$ that identifies some set of properties in the domain. For some queries, it is not necessary

to distinguish between properties $q_1, \ldots, q_n$. Therefore, we would like to replace them by one predicate q which is intended to denote the union of the relations represented by $q_1, \ldots, q_n$. The predicate q may already exist in the knowledge base (this will be the common case), or may be a newly introduced predicate. Replacing a set of predicates $q_1, \ldots, q_n$ by a predicate q has been considered as the problem of *predicate abstraction* [Plaisted, 1981; Tenenberg, 1990]. Our treatment of predicate abstraction is inspired by the work of Tenenberg. As before, we denote the result of the syntactic transformation we want to perform to a formula ϕ by $f_{\mathcal{Q}}(\phi)$. In this case, $f_{\mathcal{Q}}(\phi)$ is the result of replacing every occurrence of a predicate q_i in ϕ by the predicate q.

Our strategy is to identify formulas that are independent of $\mathcal{Q}$. Independence will be used both for defining irrelevance and for deciding to which formulas to apply $f_{\mathcal{Q}}$. As in the case of predicate arguments, the definition of independence is based on the intended semantics of the new predicate q. Specifically, suppose I is an interpretation for the set of formulas in the KB in which a predicate symbol p is mapped to a relation P. We define an abstract interpretation $Abs(I)$ for formulas in which occurrences of $q_1, \ldots, q_n$ are replaced by the predicate q. The interpretation $Abs(I)$ will have the same set of objects as I. The relations in $Abs(I)$ are $(Rel(I) - \{Q_1, \ldots, Q_n\}) \cup Q$, where $Rel(I)$ are the relations in I, and Q is a new relation.[9] The interpretation $Abs(I)$ is defined as follows:

- The interpretations of terms in I and $Abs(I)$ are identical.
- If $p \notin \{q_1, \ldots, q_n\}$, the predicate p is mapped to the same relation as in I.
- The predicate q is mapped to the union of the interpretations of $q_1, \ldots, q_n$, i.e., $Q = Q_1 \cup \ldots \cup Q_n$.

Based on the definition of abstract interpretations we define independence as follows:

Definition 6.8: A formula ψ is independent of the predicate refinement $\mathcal{Q}$, with respect to the knowledge base Δ, if for any interpretation I

$$I \models \Delta \Longrightarrow Abs(I) \models f_{\mathcal{Q}}(\psi).$$

∎

We define the abstract knowledge base resulting from removing $\mathcal{Q}$ from Δ by:

$$Abs_{\mathcal{Q}}(\Delta) = \{f_{\mathcal{Q}}(\psi) \mid \psi \in \Delta \text{ and } \psi \text{ is independent of } \mathcal{Q}\}$$

[9] If the predicate q already exists in the KB, then Q is the relation to which q is mapped

Note that if I is an interpretation, then the following holds as well:

$$I \models \Delta \Longrightarrow Abs(I) \models Abs_{\mathcal{Q}}(\Delta). \tag{6.2}$$

Returning to Example 6.2, we observe that the rule

$s : sportsCar(X) \Rightarrow highRiskToInsure(X)$

is not independent of the predicate refinement $\{sportsCar, familyCar\}$. To see this, consider the interpretation I in which $familyCar$ and car are mapped to the relation containing $\{(Camry)\}$, and both $sportsCar$ and $highRiskToInsure$ are mapped to the empty relation. The rule $f_{\mathcal{Q}}(s)$

$car(X) \Rightarrow highRiskToInsure(X)$

is not satisfied by $Abs(I)$ (which in this case contains the same interpretations for car and $highRiskToInsure$ as I).

As in the case of irrelevance of predicate arguments, weak irrelevance provides a logical justification for abstracting Δ by $Abs_{\mathcal{Q}}(\Delta)$:

Theorem 6.9: *Let $\mathcal{D}_\psi(\Delta)$ be the set of derivations of the query ψ from the knowledge base Δ. If $WI(\mathcal{Q}, \psi, \Delta, DI_1, \mathcal{D}_\psi)$ holds, then*

1. *If $\Delta \vdash \psi$, then $Abs_{\mathcal{Q}}(\Delta) \vdash f_{\mathcal{Q}}(\psi)$.*
2. *If the formula ψ does not contain predicates from $\mathcal{Q}$, then $Abs_{\mathcal{Q}}(\Delta) \models f_{\mathcal{Q}}(\psi) \Longrightarrow \Delta \models \psi$.*

The proof is given in Appendix A.

Automatically deriving Irrelevance of Predicate Refinements

As in the case of irrelevance of predicate arguments, our strategy in deriving irrelevance of predicate distinctions is to find a set of formulas Ψ that are weakly irrelevant to the query and such that the formulas in $\Delta - \Psi$ are independent of the predicate refinement. To devise such a method, we need to be able to verify that a formula is independent of the predicate refinement. Below we give a condition on clauses that enables us to verify independence.

Lemma 6.10: *A clause C is independent of the predicate refinement $\mathcal{Q}$ w.r.t. the knowledge base Δ if and only if the following condition holds.*

Suppose $Neg(C)'$ is the result of substituting every occurrence of a predicate of $\mathcal{Q}$ in $Neg(\psi)$ by some other predicate in $\mathcal{Q}$ using a mapping f_1 (two occurrences of the same predicate in $Neg(C)$ need not be mapped to the same predicate under f_1). Then there exists some C_2 such that $f_{\mathcal{Q}}(C_2) = Pos(f_{\mathcal{Q}}(C))$ and $\Delta \models C_2 \cup Neg(C)'$.[10]

[10] $C_2 \cup Neg(C)'$ denotes the clause containing the union of literals in C_2 and $Neg(C)'$.

The proof of the lemma is given in Appendix A. Note, that a clause that contains only positive literals from $\mathcal{Q}$ will be independent of the refinement whenever it is provable from Δ. In the case of Horn rules, the definition boils down to the following condition. If r is a rule (whose head is not a predicate in $\mathcal{Q}$), then it must be the case that given any mapping of occurrences of a predicates in $\mathcal{Q}$ in the antecedent of r to any other predicates in $\mathcal{Q}$, the resulting rule is still entailed by Δ.[11] For instance, in Example 6.2, we can map the occurrence of $familyCar$ in the rule $familyCar(X) \Rightarrow vehicle(X)$ to $sportsCar$ and the resulting formula will still be entailed by the knowledge base. Ground atomic formulas are independent of any predicate refinement.

Finally, note that the condition given in Lemma 6.10 involves checking whether $\Delta \models C_2 \cup Neg(C)'$, and is therefore in general undecidable. A sufficient condition can be guaranteed by checking whether $C_2 \cup Neg(C)' \in \Delta$.

6.5 Discussion and Related Work

This chapter proposes a new approach to research on reasoning with multiple levels of abstraction. At its core, the approach advocates associating an abstraction of a knowledge base with the removal of some irrelevant detail, and then using the framework for analyzing irrelevance to gain insights into the specific abstraction at hand. Specifically, the approach provides the following advantages:

1. The formal definition of irrelevance provides a logical account of the conditions under which the abstraction is appropriate.

2. The problem of automatically creating an abstraction for specific queries (based on deriving irrelevance claims) is well defined. In many cases, we can adopt existing algorithms for automatically deriving irrelevance claims in order to create abstractions.

3. The analysis of irrelevance provides insights into several properties of the abstraction, such as the utility and composibility of the abstraction.

4. Associating abstractions with irrelevance claims makes it possible to compose knowledge bases that each make certain abstractions (or simplifying assumptions) about the domain. This can be done by explicitly reasoning about the consistency and adequacy of the irrelevance claims underlying the assumptions being made by the knowledge bases.

[11] If the head of r is a predicate in $\mathcal{Q}$ then in addition to the mapping on the antecedent there must be some predicate $q_0 \in \mathcal{Q}$ such that replacing the head predicate by q_0 still yields a rule that is entailed by Δ

We have demonstrated the approach for two kinds of abstractions, removal of predicate arguments and predicate abstraction. In both cases, we have provided a logical account for the appropriateness of the abstractions, and we have developed efficient algorithms for automatically deciding which abstraction is appropriate for a given query. In the next chapter, we demonstrate how this approach can be used for automatically composing a knowledge base for a specific query from a set of knowledge base fragments that are given at multiple levels of abstraction. To pursue this approach, additional relevance subjects should be considered in detail, as well as exploring alternative definitions of irrelevance.

It should be noted that the idea of associating irrelevance with abstractions was also mentioned by Subramanian [Subramanian, 1989], but was not formalized or demonstrated concretely. Subramanian also mentions some of the new irrelevance subjects described here.

Our results on projecting predicate arguments are related to the work by Ramakrishnan et al. [Ramakrishnan *et al.*, 1988] on identifying existential queries. That work presents an algorithm for detecting cases in which arguments of subgoals in logic programs can be removed without affecting the answer to the query. Their treatment of predicate arguments differs in that they distinguish between different occurrences of a predicate in a program. As a result, their algorithm may decide to project an argument of a predicate p in one occurrence of p and not to project it in a different occurrence, thereby requiring two versions of the relation denoted by p. We can refine our treatment in the same way by applying a syntactic transformation to our knowledge base in which we rename every occurrence of a predicate p such that no two occurrences in the original knowledge base have the same predicate name. The definition of independence that we present in Section 6.3 is better motivated semantically than the one they present and applies to more than just Horn rules. The syntactic condition for independence given in Theorem 6.7 generalizes the condition given in [Ramakrishnan *et al.*, 1988] to arbitrary clauses. Finally, their algorithm for identifying irrelevant predicate arguments is based on building a rule-goal graph of the rules in the knowledge base. Our algorithms use the query-tree and can therefore detect a larger class of irrelevance claims by considering interpreted literals in the rules, minimal derivations and extended languages including negated EDB subgoals.

Our treatment of predicate abstraction in Section 6.4 is inspired by the work of Tenenberg [Tenenberg, 1990]. Tenenberg considers the problem of finding the maximal set of clauses that are independent of a predicate refinement. He presents a constructive proof for the existence of such a set and shows that unless the knowledge base Δ is empty, the set will be infinite. The abstract knowledge base we consider $Abs_{\mathcal{Q}}(\Delta)$ is a finite subset of Tenenberg's maximal set and is effectively computable in many cases. Tenenberg also considers a finite and computable subset that he calls $MembAbs$. However, $MembAbs$ is a subset of $Abs_{\mathcal{Q}}(\Delta)$. The contribution of

our work on predicate refinements is in providing a logical justification for when a predicate abstraction is appropriate for a given query and providing algorithms for automatically verifying that the justification holds.

Giunchiglia and Walsh [Giunchiglia and Walsh, 1992] present a theory of abstraction in which they identify two classes of abstractions. The first, *TD-abstractions*, requires that any formula that is derivable from the abstract knowledge base must also be derivable from the original knowledge base. The class of *TI-abstractions* requires that any formula which is derivable from the original knowledge base also be derivable from the abstract one. One of the key aspects underlying our treatment of the connection between irrelevance and abstractions is the intuition that removing irrelevant knowledge should not result in the ability to derive conclusions that were not derivable earlier. As a result, in the examples we presented, the abstractions justified by irrelevance claims fall under the class of TD-abstractions. Moreover, weak irrelevance also guarantees that if the query was derivable in the original knowledge base, it will also be derivable in the abstract knowledge base. Therefore, our abstractions can be viewed as being TI-abstractions with respect to a specific query.

Historically, TI-abstractions have received more attention (e.g., [Sacerdoti, 1974; Plaisted, 1981]). In that work, the intuition was that in most cases the information removed was irrelevant to the query, and therefore the answer obtained from the abstract knowledge base would hold (or could be refined to an answer) in the original knowledge base. For example, ABSTRIPS [Sacerdoti, 1974] made the assumption that the action preconditions of lower criticality values are easier to achieve and can therefore be ignored when formulating an abstract plan. The utility of the abstraction depended on how often the problem solver would have to backtrack across abstraction levels. To articulate the intuition behind these kinds of abstractions by irrelevance claims, we need to extend our framework for reasoning about irrelevance to include probabilistic (or default) irrelevance claims. We need to be able to state that some subject is irrelevant to a query with some probability (or under some conditions). To understand these abstractions we also need to refine the theory of Giunchiglia and Walsh. Their theory is based on two kinds of relationships between the statements:

S1. $\Delta \vdash q$ and

S2. $Abs(\Delta) \vdash Abs(q)$.

where $Abs(\Delta)$ and $Abs(q)$ are the abstractions of the knowledge base and the query respectively. For TD-abstractions, they require $S2 \Rightarrow S1$, while for TI-abstractions they require $S1 \Rightarrow S2$. Instead of considering only these two strict relationships, we can consider other possibilities. For example, we can require that if S2 holds, then there is some prespecified condition on q and the possible derivations of q such that S1 holds. Such a condition should be useful in telling us whether the answers given

from the abstract knowledge base would follow from the original knowledge base, or how to refine a derivation in the abstract KB into a derivation in the original KB.

Our analysis of irrelevance of predicate arguments is one instance of this more general class. For example, if the first argument of a binary predicate p is irrelevant to a query, and $p(a)$ is derivable from $Abs(\Delta)$, then this guarantees that there exists some X such that $p(X, a)$ is derivable from Δ.

Knoblock's ALPINE system [Knoblock, 1990] is another example of this generalization. The ALPINE planner creates an abstraction hierarchy which guarantees that if there is a plan for a goal in the original problem space, then there will be one in the abstract space such that the original plan is a *monotonic refinment* of the abstract plan. This condition enables the planner to considerably prune its search when it refines an abstract solution, since it need only consider monotonic refinements of the abstract plan. Knoblock et al. [Knoblock *et al.*, 1991] present other examples of possible relationships between abstract and concrete plans which are then used to prune the search of a planner.

Additional work on automatically generating abstractions is described in [Ellman, 1990; Ellman, 1992; Lowry, 1992]. Work on analysis of the utility of abstractions is described in [Knoblock, 1990; Bacchus and Yang, 1992].

Chapter 7

Automated Modeling of Physical Systems

The previous chapter described how relevance reasoning can play a key role in facilitating reasoning in complex domains that require extensive use of abstractions. An important domain with such characteristics is that of modeling of physical systems. In this domain, we are given a theory of the physical world, a description of a specific system and a query about that system. Our goal is to choose a representation for the system that will enable us to answer the query effectively. Physical systems can be represented in multiple ways, using several levels of detail, abstraction and differing perspectives. Therefore, the main challenge in solving this problem is choosing among alternative possible representations of the system. The chosen representation must be adequate for answering the query, but must also be as parsimonious as possible, in order to allow efficient inference.

This chapter considers the automated modeling problem from the perspective of relevance reasoning. In doing so, we shed light on the problem, showing that certain aspects of it can be automated by simple considerations of relevance reasoning. Furthermore, we show how additional domain knowledge, which is needed for model selection, can be expressed in the form of irrelevance claims. We combine these observations into a novel model selection algorithm, based on relevance reasoning. Section 7.1 describes the model formulation problem and its relation to relevance reasoning. Section 7.2 describes our algorithm and Section 7.3 presents an analysis of its properties.

7.1 Problem Formulation

In order to reason about a physical system for tasks such as simulation, design or diagnosis, we need some representation of the system. We call such a representation

a *model* for the system. In this chapter, a model refers to a representation of a system. We use the phrase *logical-model* to refer to the concept of a model in Mathematical Logic (cf. [Enderton, 1972]).

For complex physical systems, there is typically no single model of the system that will be adequate and enable efficient inference for all possible queries. Consequently, the goal of the automated modeling problem is to find a model for a system that is best suited for a specific query.

7.1.1 Compositional Modeling

We construct a model for a given physical system based on the *Compositional Modeling* approach described in [Falkenhainer and Forbus, 1991]. In this approach, a physical situation is modeled as a collection of *model fragments*. Each model fragment represents some atomic aspect of a physical object or a physical phenomenon. For example, a model fragment may describe the dependence of the voltage of a battery on its charge level (as shown in Figure 7.1), or it may describe the process of fluid flow through a pipe connecting two containers.

A model fragment contains a set of *participants*, which are the set of objects in the domain that are taking part in the phenomenon being described. An *instantiated* model fragment is a binding of each of the participants to an object in the domain. The model fragment contains a set of *operating_conditions* which the participants need to satisfy in order for the instantiation to be valid. The *behavior_conditions* of the model fragment specify the behavior of the participant objects in the phenomenon being modeled.

A model for a system in a given state is a set of instantiated model fragments whose operating conditions are satisfied. The union of the behavior conditions of the instantiated model fragments gives rise to a *simulation model* for that state. The simulation model is used to determine the next state of the system, in which a new simulation model is chosen.

A model fragment consists of the following components:[1]

Participants: These are the set of objects participating in a model fragment instance. A participant can be viewed as a unary function from a model fragment instance to the objects of the domain. In Figure 7.1, the participant is an instance of class `battery`.

Variables: These are time dependent variables associated with the participants in a model fragment instance. We distinguish two kinds of variables. The first, which are also called *quantities*, are variables that are continuous over time (e.g.,

[1] For a complete formal discussion of model fragments, see [Farquhar *et al.*, 1993]. The description below includes only the aspects relevant to our discussion

Charge-sensitive-voltage (X : battery)

Variables:

```
voltage(X), chargeLevel(X), damaged(X)
```

Operating conditions:

```
¬damaged(X)
6 < charge-level(X) < 30
```

Modeling conditions:

```
Relevant(rechargeableBattery(X)) ∧
Relevant(chargeLevel(X)) ∧
Relevant(voltage(X))
```

Behavior conditions:

```
voltage(X) = f(chargeLevel(X))
```

Figure 7.1: An example model fragment

voltage, current). The second kind are binary variables that may change over time (e.g., *damaged*(*battery*), *on*(*switch*)). Binary variables are represented by ground atomic literals.[2] In our example, the quantities are the voltage and charge level of the battery and *damaged*(*X*) is a binary variable.

Operating conditions: These conditions specify when an instance of the model fragment exists. They are conditions on the participants of the model fragment and on its variables. They include both structural constraints on the participants as well as constraints on the ranges of the variables. In our example, we require that the battery not be damaged and that the charge level of the battery be between 6 and 30.

Modeling conditions: These are conditions on the model of the system that need to be satisfied in order for an instance of the model fragment to exist. They are used in order to distinguish different ways of modeling the same phenomenon. We distinguish two classes of modeling conditions. The first class consists of relevance claims. As explained in Chapter 6, relevance claims can be used to express the assumptions underlying an abstraction. For example, a description of the battery that ignores its thermal aspects may be based on the irrelevance claim stating that the predicate *temperature* is irrelevant to the query. We assume that all the irrelevance claims used in the modeling conditions are based on a single definition of irrelevance in our space. In the modeling conditions we use

[2] Note that the time argument of all variables is left implicit.

the predicate *relevant*, which should be thought of as denoting the complement of *irrelevant*. The second class of modeling constraints includes assumptions about the problem solving task. These include assumptions about the desired accuracy of the answer and the temporal granularity of the model (e.g., we will model a battery differently depending on whether we are considering its behavior over one second or over one year).

In our discussion we assume the following convention about the interpretation of predicates used in the modeling conditions. A positive literal of a predicate is assumed to denote an assumption that yields a more complicated model of a phenomenon. A negative literal denotes a *simplifying* assumption. For example, $\neg relevant(Temperature(battery))$ states that the representation is simplified to ignore the thermal aspect of the battery, whereas the literal $relevant(Temperature(battery))$ in the modeling conditions states that the model fragment considers the temperature aspect of the battery. As another example, the literal $\neg large(timeScale)$ states that the representation is simplified to ignore longer term effects on the battery.[3]

In our example, the model fragment requires that the charge level and the voltage are considered relevant properties, and that the rechargeability aspect of the battery be relevant as well. Modeling conditions are distinguished from operating conditions in that they are conditions about the model (i.e., meta-level conditions) as opposed to conditions on the domain and state.

Behavior conditions: The statements in the behavior conditions are true whenever the instance of the model fragment exists. Essentially, these sentences describe the phenomenon being modeled. We distinguish three kinds of behavior conditions. The first kind describe continuous phenomena (e.g., a fluid flow) by a set of equations involving the continuous quantities of the model fragment. The equations may be quantitative (algebraic and ordinary differential equations) and can also be qualitative (e.g., the rate of evaporation negatively affects the amount of water in the cup). The second kind of behavior conditions describe instantaneous changes of the binary variables of the model fragment (e.g., turning off a switch). Finally, the third kind of behavior conditions describe time independent properties of participants in the model fragment. We assume that a model fragment contains behavior conditions of only one kind. In our discussion we assume that the behavior conditions do not contain inequalities on

[3]Some assumptions may be multivalued. For example, the time scale may be either `small`, `medium` or `large`. The algorithms we describe in this chapter can be extended in a straightforward fashion to deal with such assumptions. However, for clarity we assume here that modeling assumptions are binary.

quantities. The behavior conditions of the model fragment in our example describe the functional relationship between the voltage and the charge level of a battery.

The semantics of model fragments can be summarized as follows. Let $f_1, \ldots, f_n$ be the participants of a model fragment M. Let $o(X_1, \ldots, X_n)$ be its operating conditions, $a(X_1, \ldots, X_n)$ be the modeling conditions, and $b(X_1, \ldots, X_n)$ be its behavior conditions. First, whenever a set of objects satisfies the operating and modeling conditions, there exists an instance of the model fragment. Formally:

$$\forall X_1, \ldots, X_n \left[(o(X_1, \ldots, X_n) \wedge a(X_1, \ldots, X_n)) \Leftrightarrow (\exists\, m) M(m) \wedge \bigwedge_{j=1}^{n} f_j(m) = X_j \right]$$

The existence of the model fragment also implies that the variables mentioned in it are defined. Furthermore, the existence of the model fragment implies that the behavior conditions hold:

$$\forall X_1, \ldots, X_n \left[(\exists\, m) M(m) \wedge \left(\bigwedge_{j=1}^{n} f_j(m) = X_j\right) \Rightarrow b(X_1, \ldots, X_n) \right].$$

Composing Simulation Models

Given a description of the physical configuration of a system, a particular state it is in, and a query about the state, the task is to formulate a model that represents the physical phenomena occurring in the state. Such a representation is composed of a set of instantiated model fragments whose operating conditions and modeling conditions are satisfied in that state. These model fragment instances are called the set of *active* model fragments in that state, and together will comprise the *simulation model* for that state. The behavior conditions of the model fragments in the simulation model give rise to a set of equations and logical formulas that must hold among participants and the variables as a consequence of the phenomena taking place. They are used to determine the next state of the system in which a new simulation model is selected.

The main advantage of compositional modeling that makes it appropriate for our task is its modularity. Writing model fragments, each describing a single phenomenon, is a much easier task than composing a complete model for every possible system and query. Adding model fragments to an existing library is also much easier. Furthermore, model fragments can be reused in any appropriate context.

7.1.2 The Model Fragment Library

To facilitate compositional modeling, we impose additional structure on the model fragment library. Specifically, model fragments are grouped into *composite model fragments* (CMFs), and CMFs are further grouped into *assumption classes*. Before we discuss these constructs, we briefly describe the notion of causal ordering of quantities.

Causal Ordering

Equations in model fragments describe the relationships among the continuous variables involved in the modeled phenomenon. These relations have no causal import. For example, the equation for Ohm's law, $V = iR$, only states the relationship between the current, the voltage and the resistance. In building a model for a system and explaining it, we often want to know exactly how the variables are determined, i.e., what are the causal dependencies between the quantities in the model. For example, in a model containing Ohm's law, we may say that the voltage is determined by the current and the resistance. A *causal ordering* [Iwasaki and Simon, 1986; de Kleer and Brown, 1986] specifies the dependency structure among the quantities in the model fragment.[4] It is specified by causally orienting every equation in the model fragment, i.e., associating one quantity $f(e)$ with every equation e in the model. The quantity $f(e)$ must be part of e, and must not be associated with any other equation in the model fragment (i.e., if $e_1 \neq e_2$ then $f(e_1) \neq f(e_2)$). The quantities in the model fragment that are not associated with any equation are called *exogenous* in the causal ordering f. The exogenous quantities are assumed to be determined by other phenomena (described by other model fragments), and can therefore be considered as input to the current model fragment. Given a causal ordering f, we say that a quantity v_1 causally affects a quantity v_2 if:

- The quantity v_1 appears in the equation e, and $f(e) = v_2$, or
- There exists some quantity v_3, such that v_1 causally affects v_3 and v_3 causally affects v_2.

If v is a continuous variable, we say that a model fragment m can determine it if there is some causal ordering of the quantities in m such that v is not exogenous. If v is a binary variable, we say that it can be determined by m if it appears in its behavior conditions.

The set of variables that can be determined by a model fragment are called its *output variables*. We now describe the structures in the model fragment library.

Composite Model Fragments

Some model fragments describe the same phenomenon, but differ only in their operating regions, i.e., the value ranges assumed for the continuous variables in the model fragment. For example, the function describing the dependence of the voltage of a battery on its charge level changes depending on the value of the charge level:

```
chargeLevel ≤ 6 ⇒ voltage = f1(chargeLevel)
```

[4] We do not consider causal dependencies among binary variables here.

```
6 < chargeLevel < 30 ⇒ voltage = f2(chargeLevel)
chargeLevel ≥ 30 ⇒ voltage = f3(chargeLevel)
```

In selecting a model that will be adequate for multiple states of a simulation, it is easier to think of such model fragments as being grouped into a single *composite model fragment* (CMF). A CMF is a set of model fragments describing the entire operating range of the variables participating in the phenomenon. In every state of the simulation, the operating conditions will guarantee that only one model fragment from every CMF will be included in the simulation model. Clearly, a CMF can also be a singleton set.

Assumption Classes

Composite model fragments are further grouped into assumption-classes.[5] An assumption class is a set of CMFs that describe the same phenomenon based on different and contradictory modeling conditions. As stated, modeling conditions express the assumptions that we are making in the representation of the system. They express the underlying abstractions and approximations that are assumed by the model fragment. Figure 7.2 shows an assumption class consisting of different ways of describing the voltage of the battery. One way to model the voltage is to assume it is constant. Another way is to assume it degrades over time. More complicated ways of modeling the battery consider aspects such as the charge level and the temperature. Since CMFs in an assumption-class are contradictory, any consistent set of modeling assumptions will include at most one CMF from a single instantiated assumption class.

CMFs in an assumption class are partially ordered by a *simplicity* relation, denoted by the predicate $\prec$. A CMF c_i is said to be simpler than a CMF c_j if c_i makes a superset of the simplifying assumptions made by c_j. The transitive closure of $\prec$ will be denoted by the predicate $\prec^*$. In the figure, the simplicity relation is denoted by the directed arcs. We assume that every assumption class has a single most complicated CMF and a single simplest CMF. The former represents the most detailed way of describing a phenomenon, while the latter represents the simplest way of doing so (e.g., the voltage of the battery can be modeled as constant). Finally, we assume that if $c_i \prec c_j$ then:

1. The output variables of c_j are a superset of the output variables of c_i.
2. If f_i is a causal ordering of the variables of c_i, then there exists a causal ordering f_j of c_j such that the causal relations among variables in c_i (given by f_i) are a subset of the causal relations among variables in c_j (given by f_j).

[5] The term assumption-class is used in order to be consistent with [Falkenhainer and Forbus, 1991], not because it is especially appropriate

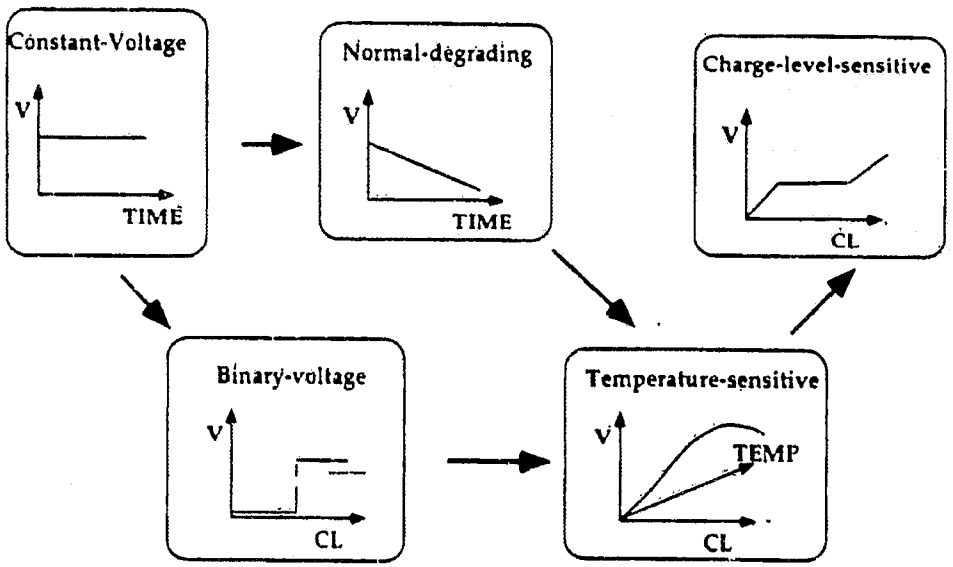

Figure 7.2: Battery voltage assumption class

All of these properties follow if we assume that whenever $c_i < c_j$, then c_i is a *causal approximation* of c_j [Nayak, 1992b]. Nayak has shown that causal approximations cover most approximation relations encountered in practice.

In contrast to previous treatments of assumption classes, we assume that the modeling conditions of CMFs in an assumption class *precisely* characterize the differences of assumptions made by CMFs in the assumption class. Specifically, this is formalized as follows. Suppose the modeling conditions of a CMF c is the conjunction of the literals in the set As_c, and suppose $c_i < c_j$. Then we can annotate the link from c_i to c_j with a set of positive literals $p_1, \ldots, p_n$, which means that c_i is making the simplifying assumptions $\{\neg p_1, \ldots, \neg p_n\}$ in addition to the simplifying assumptions made by c_j, or formally,

$$As_{c_i} = (As_{c_j} \setminus \{p_1, \ldots, p_n\}) \cup \{\neg p_1, \ldots, \neg p_n\}.$$

The articulation of these differences will play an important role in selecting the simplest model.

Other Assumptions About the Library

Composing non-contradictory model fragments: A variable may be affected by more than one phenomenon and therefore by more than one assumption class. For example, the amount of water in a container can be affected by an evaporation process and by a condensation process. These two phenomena are represented by different assumption classes. One of the key assumptions on model fragments in the compositional modeling approach is that they be *composable*. Specifically, this means the following. Let m_1 and m_2 be two model fragments that have consistent operating conditions and modeling conditions, such that both can determine a variable v. Then, m_1 and m_2 can be composed to a single model fragment m_3 describing the union of the phenomena in m_1 and m_2. The procedure for creating m_3 is assumed to be given.

Coherence of the Library

The *library coherence* assumption essentially requires that if we have a set of model fragments that have consistent modeling assumptions and whose operating conditions are satisfied, then the resulting set of equations will not be over-constrained (i.e., will not have more equations that quantities). Formally, this assumption is defined as follows:

Definition 7.1: A model fragment library satisfies the library coherence assumption if the following condition holds. Let M be any set of model fragments in the library and s be any state such that:

1. The conjunction of modeling conditions of model fragments in M, $a(M)$, are consistent.
2. If $a(M) \models Relevant(v_1)$, then v_1 appears in some model fragment in M.
3. The conjunction of the operating conditions of model fragments in M are satisfied in s.

Then, the set of equations given by the union of the behavior conditions of M are not over constrained. ∎

Note that a set of equations that are not over constrained can always be made complete by assuming that some variables are exogenous.

7.1.3 Other Modeling Constraints

Except for the modeling conditions attached to each model fragment, we assume a background theory of modeling constraints, $\mathcal{C}$. We use $\mathcal{C}$ to express additional

constraints on the possible models. The constraints can either be domain independent (e.g., general constraints entailed by relevance claims) or domain specific constraints. For example, the following constraint states that if the refinement of objects along the property r is relevant for an object o that is relevant to the query, and $r(o, X)$ holds, then X is relevant to the query:[6]

$$relevantObjectRefinement(O, R, G) \wedge r(O, X) \wedge relevantObject(O, G) \Rightarrow relevantObject(X, G).$$

The constraints in C may also be heuristic in nature. For example, the following constraints are a variation on the *object expansion* heuristic used in [Falkenhainer and Forbus, 1991]. They are used to enforce the relevance of certain objects in a system, given the initial set of relevant objects.

$$structuralHierarchySlot(p) \wedge relevantObject(X, G) \wedge relevantObjectRefinement(X, P, G) \wedge p(X, Y) \Rightarrow relevantObject(Y, G)$$

$$structuralHierarchySlot(p) \wedge p(X, Y) \wedge relevantObject(X, G) \wedge relevantObject(Y, G) \Rightarrow relevantObjectRefinement(X, p)$$

The heuristic states that if the objects s_1 and s_2 are both relevant to the query, and t is their least common ancestor in the structural hierarchy, then any object in the hierarchy that is either in between t and s_1 (or between t and s_2), or a child of such a object, will be considered relevant to the query.

Essentially, constraints can be expressed using arbitrary first order formulas. For efficiency reasons, we assume that the constraints in C are expressed using only Horn rules. In practice, Horn rules have been expressive enough for the modeling constraints we have encountered. —

7.1.4 The Model Formulation Problem

Informally, the model formulation problem is to choose a simulation model (i.e., a set of instantiated model fragments) that can answer a given query about a system in a specific state. However, a simulation of a system may go through many states, and we do not want to repeat the costly selection process at every state. Therefore, we pose the model formulation problem as selecting a small set of CMFs, called the *scenario model*. The scenario model has the property that its modeling conditions are consistent, and that at every state, we can choose a simulation model from it easily.

Formally, the model formulation problem is to choose a scenario model, given the domain theory (i.e., model fragment library and background modeling constraints), a system description and a query, defined as follows:

[6] Note that we use specific predicate names in order to make the type of the subject in the relevance claim explicit

- **System description:** A set of facts about a physical system and its initial state. This description typically includes a set of individuals (i.e., components of the system), their physical structure and the initial values of variables in the system.
- **Query:**
  - A variable v (or list of variables) whose behavior we want to predict in the simulation of the system.
  - A list E of exogenous variables and terms (i.e., ground atomic formulas). The elements in E are assumed to be given and are outside the scope of the simulation for which we are constructing a scenario model. We can use E to circumscribe the set of states for which we are creating a scenario model (e.g., we may construct a scenario model only for states in which the battery is not damaged).
  - A list *Init* of modeling constraints that we want to enforce. Implicitly, *Init* includes $Relevant(v)$.[7]

A scenario model is a set of instantiated CMFs whose modeling conditions are consistent. At every state the system checks the operating conditions *only* of the CMFs in the scenario model. The conditions of at most one model fragment from each CMF will be satisfied in the state, and these model fragments will comprise the simulation model of the state. We denote the scenario model by $\mathcal{S}$ and the simulation model created from it in state s by $\mathcal{S}_s$.

The resulting scenario model must satisfy several properties. First, it must be adequate for answering the query. This means that it must be coherent and sufficient as follows:

Definition 7.2: A scenario model $\mathcal{S}$ is adequate if

C1. There is a logical-model M for the background constraints $\mathcal{C}$ such that

  A. All the modeling conditions of CMFs in $\mathcal{S}$ are satisfied in M.

  B. If $Relevant(v_1)$ is satisfied in M and v_1 is a variable, then some CMF in $\mathcal{S}$ includes v_1.

C2. For any state s of the simulation, the equations arising in $\mathcal{S}_s$ can be made *complete* (i.e., not over-constrained or under constrained) by adding exogenous variables. Furthermore, the equations in $\mathcal{S}_s$ include the variable v, and v is not exogenous in the complete set (and therefore we can say that $\mathcal{S}_s$ determines v).

[7] Note that these constraints can also be specified as part of $\mathcal{C}$. However, it is often more natural to specify them as part of the query

In order for a scenario model to be useful, it should be as simple as possible:

Definition 7.3: A scenario model $\mathcal{S}_1$ is simpler than $\mathcal{S}_2$ if there is a mapping $\phi : \mathcal{S}_1 \rightarrow \mathcal{S}_2$ on the CMFs of $\mathcal{S}_1$, such that

1. For every $c \in \mathcal{S}_1$, $\phi(c)$ is from the same instantiated assumption class as c.
2. Either $c = \phi(c)$ or $c <^* \phi(c)$.

■

The model selection problem is to find a scenario model that is adequate and such that there is no simpler adequate scenario model.

7.1.5 Model Formulation as Relevance Reasoning

The approach to the model formulation problem advocated in this chapter is based on the intuition that several aspects of the problem can be viewed as relevance reasoning. We explain our view in this section.

Intuitively, the model formulation problem can be viewed as a combination of two subproblems. The first is to determine which phenomena (and therefore which variables) are relevant to the query variable. The second problem is to determine the level of detail at which to model the relevant phenomena. These problems are closely related, because the decision to model a certain phenomenon at a greater level of detail may require modeling additional phenomena.

Selecting the Relevant Phenomena

The first part of the model formulation problem is to decide which variables are relevant to the query variable (and therefore, decide which phenomena should be modeled). Intuitively, a variable u is relevant to the query variable v if u can *causally influence* v, i.e., either (1) there is some state of the system in which u causally affects v or (2) u can cause a change in the state of the system (and therefore indirectly affect the value of v). Consequently, finding the relevant variables can be done by following the possible causal influences between variables. The algorithm that we describe in this chapter traces through all the possible causal influences on the query variable. Note that the intuition underlying this algorithm is similar to the intuition underlying the construction of the query-tree, where we represented all the possible derivations of the query.

Selecting the Level of Detail

The second part of the model selection problem is determining the level of detail at which to model each phenomenon. This entails deciding which abstractions and approximations can be made in modeling the system. As described in Chapter 6, knowledge underlying such decisions can be stated as relevance claims and better understood when stated in that form. In our algorithm, we bring relevance knowledge to bear in two ways:

- We articulate the *difference* between CMFs in an assumption class by the modeling constraints, expressed partially by relevance claims. Previous treatments of assumption classes require that every CMF have a set of modeling constraints, but they do not require that the constraints be related in any way (except for being mutually exclusive). Articulating the precise differences between the CMFs is a more principled method of building assumption classes and enables us to determine when to switch from one model fragment to another.

- Engineers have good general heuristics for selecting relevant detail in modeling of physical systems. We use the modeling constraints C to express these heuristics declaratively and reason with them.

Our modeling algorithm will use both kinds of this knowledge to select the simplest scenario model.

Partial Knowledge about the Simulation States

Our algorithm selects a scenario model for a set of possible states of the system. Envisioning all the possible states that the system may reach beginning from the initial state is a very expensive operation [de Kleer and Brown, 1984], which we do not want to perform as part of the model formulation process. Therefore, our algorithm selects the scenario model based only on partial knowledge of the possible states. This knowledge is given implicitly by the set E of the variables that are assumed to be exogenous and the time invariant facts in the description of the system. The problem we face here is analogous to the relevance reasoning problem considered in Chapter 3, in which we wanted to decide which ground formulas are irrelevant to the query without actually knowing the contents of the database. In analogy to what we did there, this entails that we assume that the system can actually reach any state that is consistent with our partial knowledge. As in Chapter 3, any additional knowledge about the reachable states may enable us to select a simpler scenario model. Assuming partial knowledge about the world is a key aspect in making relevance reasoning practical.

7.2 Model Formulation Algorithm

Based on these observations, we now describe our model formulation algorithm. Informally, the algorithm follows all possible influences on the query in order to find all the variables that can affect the query. For each such variable, the algorithm selects the simplest CMF that describes it such that the set of selected CMFs make consistent modeling assumptions.

To find all the variables that can affect the query, the algorithm begins by considering all the assumption classes in which the query variable may be an output variable. From each such assumption class we select one CMF and recursively consider all the variables that can affect the query through the chosen CMF. These include:

1. If v is a quantity, we include all the quantities that appear in the same equation with v.

2. All the variables that appear in the operating conditions of the model fragment.

The recursion bottoms out when we reach the exogenous variables given in E.

To select a CMF from an assumption class, we maintain a list, *Rel*, of modeling assumptions made thus far about the model. The list initially includes the assumptions given in *Init* (and in particular, the relevance of the query). At every step, we choose the simplest CMF that does not contradict the assumptions in *Rel*.

Adding a new CMF to the scenario model may imply that we add additional assumptions to *Rel*, and that we need to revise previous choices of CMFs. We perform adjustment steps (via the while-loop in **select-scenario-model**) until all the choices of CMFs are consistent. The details of the algorithm are shown in Figure 7.3. Note that $Pos(As_c)$ denotes the positive literals in the assumptions made by a CMF c. The function $DeductiveClosure(D)$ returns the set of ground atomic formulas derivable from D and the rules in $\mathcal{C}$. In what follows, we illustrate the execution of the algorithm with an example.

Example

The example is a simple circuit containing a solar array (`SA1`) and a rechargeable battery (`BA1`), shown in Figure 7.4. Figure 7.5 shows the scenario description and Figure 7.6 shows the model fragments in the library. For each CMF in the domain theory, the CMF's behavior conditions and the list of variables appearing in its operating conditions are shown. The annotated assumption classes are shown in Figure 7.7. The query is `Voltage(BA1)`, with a list of exogenous variables which includes all the variables mentioned in the scenario description except `Damaged(BA1)`. The set of modeling constraints is empty.

```
procedure select-scenario-model(v, E, Init, C)
begin
   Q = {v}.
   Rel = Init.
   Model = nil.
   repeat
      q = dequeue(Q).
      As = assumption classes in which q can be an output variable and
           whose operating conditions do not contradict E.
      for each a ∈ As do:
           select-from-assumption-class (a, q).
           while there is a pair (c, q') ∈ Model such that ¬p ∈ As_c and p ∈ Rel
                remove (c, q') from Model.
                select-from-assumption-class (A_c, q').
                /* A_c is the assumption class from which c was chosen */
   until Q is empty.
   return the set {c | (c, q') ∈ Model}.
end select-scenario-model.

procedure select-from-assumption-class ( A, q )
/* A is an instantiated assumption class determining q. */
begin
   c = The simplest CMF in A such that ∄ p(¬p ∈ As_c ∧ p ∈ Rel).
   Model = Model ∪ {(c, q)}.
   Rel = DeductiveClosure(C ∪ Rel ∪ Pos(As_c)).
   inputs = the union of:
      The quantities that appear in equations with q and
      The terms in the operating conditions of c.
   for every X ∈ inputs do
      if X has not been in Q and X ∉ E then
        enqueue X onto Q.
   if relevant(q_1) ∈ Rel and q_1 ∉ E and q_1 has not been in Q then
        enqueue q_1 onto Q.
end select-from-assumption-class.
```

Figure 7.3: Model selection algorithm.

The query variable `Voltage(BA1)` is the only item on the queue initially, and so we identify `Battery-voltage-ac(BA1)` as an assumption class that can affect it.[8] To select a CMF out of this assumption class, we start from the simplest, `Constant-voltage-CMF`. Since there are no earlier modeling assumptions, this choice is consistent, and we select this CMF. This results in addition of the following to our modeling

[8] We assume that there is a data structure that enables us to efficiently find the assumption classes that affect a given variable without searching the whole model fragment library.

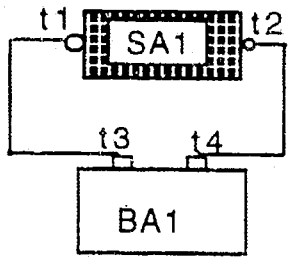

Figure 7.4: An example circuit. SA1 is a solar array and BA1 is a rechargeable battery.

Scenario Description	Legend
Solar-array(SA1)	CL: Charge-level(X)
Battery(BA1)	V: Voltage-produced(X)
Rechargeable(BA1)	TEMP: Temperature-of(X)
Plus-terminal(BA1)=t4	I: Current(Plus-terminal(X))
Minus-terminal(BA1)=t3	DOD: Average-depth-of-discharge(X)
Plus-terminal(SA1)=t2	TSLC: Time-since-last-conditioning(X)
Minus-terminal(SA1)=t1	
Electrically-connected(t2,t4)	
Electrically-connected(t1,t3)	
¬Damaged(BA1).	

Figure 7.5: The initial state of the system

assumption list, *Rel*:
`Relevant(Battery(BA1))` and `Relevant(Damaged(BA1))`.

Since the variable `Voltage(BA1)` can be influenced by the variable `Damaged(BA1)` (through the CME `Constant-voltage-CMF(BA1)`) which is not exogenous, the variable `Damaged(BA1)` is placed on the queue and becomes the new current goal. We find the assumption class `Battery-damage-due-to-overcharge-ac` that can affect `Damaged(BA1)`, out of which `Battery-damage-CMF` is selected since it is the only member. This selection causes the literals:

`Relevant(Rechargeable(BA1))` and
`Relevant(Charge-level(BA1))`

to be added to *Rel*. However, this makes the assumption list inconsistent since both ¬`Relevant(Rechargeable(BA1))` and ¬`Relevant(Charge-level(BA1))` were assumed by `Constant-voltage-CMF`.

CMF	Behavior	Variables in Operating Cond.
Battery-voltage assumption class:		
Constant-voltage-CMF	$V = C_0$	Battery(X), ¬Damaged(X)
Binary-voltage-CMF	$V = \begin{cases} 0 & \text{if } CL < c_0 \\ v_1 & \text{if } CL \geq c_0 \end{cases}$	Battery(X), ¬Damaged(X)
Normal-degrading-CMF	$V = f(Time)$	Battery(X), ¬Damaged(X)
Charge-sensitive-CMF	$V = f(CL)$	Battery(X), ¬Damaged(X), Rechargeable(X)
Temperature-sensitive-CMF	$V = f(TEMP, CL)$	Battery(X), ¬Damaged(X), Rechargeable(X)
Battery-charge-level assumption class:		
Constant-charge-level-CMF	$CL = c_1$	Battery(X), ¬Damaged(X)
Normal-accumulation-CMF	$CL = Idt$	Battery(X), ¬Damaged(X), Rechargeable(X)
Accumulation-with-aging-CMF	$CL = Idt + f(DOD, TSLC)$	Battery(X), ¬Damaged(X), Rechargeable(X)
Battery-damaged-due-to-overcharge assumption class:		
Battery-damage-CMF	$Damaged(X)$	Battery(X), ¬Damaged(X), Rechargeable(X) CL(X),

Figure 7.6: Scenario description and model fragments

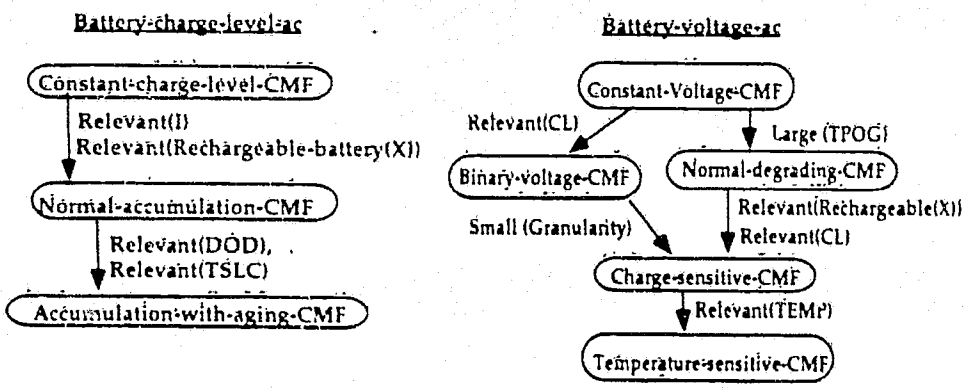

Figure 7.7: Assumption Classes

To resolve the inconsistency, we adjust the choice of `Constant-voltage-CMF`, and we now select `Charge-sensitive-CMF`, which is the simplest CMF that does not contradict the current modeling assumptions.

The current goal variable now becomes `Charge-level(BA1)`. The assumption class that can affect this variable is `Battery-charge-level-ac`, and we select from it the CMF `Normal-accumulation-CMF(BA1)`, which is the simplest CMF that is consistent with the current modeling assumptions. `Current(Plus-terminal(BA1))` can influence `Charge-level(BA1)` through this CMF. However, since it is an exogenous variable, it is not placed on the queue. The queue is now empty and the procedure

terminates. The resulting scenario model contains:

```
Charge-sensitive-CMF(BA1),
Battery-damage-due-to-overcharge-CMF(BA1),
Normal-accumulation-CMF(BA1).
```

and makes the following modeling assumptions:

```
Relevant(Battery(BA1)), Relevant(Damaged(BA1)),
Relevant(Rechargeable(BA1)), Relevant(Charge-level(BA1)),
Large(TPOG), Small(Granularity),
¬Relevant(Temperature(BA1)),¬Relevant(DOD),¬Relevant(TSLC)
```

Note that the procedure terminates at this point in the example because the variable `Current(Plus-terminal(BA1))` was specified as exogenous in the query. Had it not been specified exogenous, the procedure would have added more CMFs to the model, including those representing other components and processes affecting the current.

7.3 Analysis

In this section we prove that our algorithm produces the simplest adequate scenario model for answering the query. We discuss the assumptions under which this result holds and discuss the consequences of relaxing them. The following theorem establishes the main properties of the algorithm:

Theorem 7.4: *Let $\mathcal{M}$ be a library of model fragments describing the domain, and C be a set of modeling constraints. Let S be a description of a system and $(v, E, Init)$ be a query about the system. Let $\hat{S}$ be the scenario model resulting from algorithm* **select-scenario-model**. *Furthermore, assume that:*

- *The library coherence assumption holds.*
- *If c_i and c_j are two CMFs in an assumption class, such that $c_i < c_j$, then c_i is a causal approximation of c_j.*
- *All modeling constraints in C are either ground atomic formulas or Horn rules.*
- *The most complicated scenario model, defined to be all the possible instantiations of CMFs that are the most complicated in their assumption class, is adequate for answering the query.*[9]

[9]Note that the most complicated scenario model needs to include only the *valid* instantiations of model fragments, i.e., instantiations in which the objects satisfy the type conditions in the definition of the model fragment and the time invariant facts in the description of the system.

Then, $\mathcal{S}$ is an adequate scenario model for $(v, E, Init)$ and there is no scenario model $\mathcal{S}_1$ such that $\mathcal{S}_1$ is simpler than $\mathcal{S}$.

Proof: First we note that because of the last assumption in the statement of the theorem, the algorithm (i.e., the while loop in **select-scenario-model**) must terminate. This is because we can always adjust a choice of a CMF to a more complicated CMF that does not contradict the assumptions in Rel. Ultimately, we will end up with the most complicated scenario model which is guaranteed to be adequate.

We first consider the adequacy of $\mathcal{S}$. Condition C1 of adequacy requires that there exist a logical-model M of the modeling constraints C in which all the modeling assumptions of the CMFs in $\mathcal{S}$ are satisfied (condition A) and such that any variable v for which $Relevant(v)$ is satisfied in M is included in one of the CMFs in $\mathcal{S}$ (condition B). We define M to be the logical-model which satisfies the positive literals in Rel, and the negation of positive literals not appearing in Rel. M is a logical-model of C because $Rel \cup C$ is closed under deduction. Condition A is satisfied because the following holds in the algorithm:

(1) For every CMF $c \in \mathcal{S}$, $Pos(As_c) \subseteq Rel$.

(2) Whenever some simplifying assumption of a CMF c is not satisfied in Rel (i.e., $\neg p \in As_c$ but $p \in Rel$), we adjust the choice of c.

Because of (1), all the positive literals in c are satisfied in M. Because of (2), all the negative literals of c are satisfied in M.

Condition B is satisfied by M because whenever $Relevant(v_1) \in Rel$, where v_1 is a variable, then either

- $v_1 \in E$, or
- v_1 was put on the queue, and some assumption class that can affect v_1 was subsequently added to $\mathcal{S}$.

Therefore, in both cases, $\mathcal{S}$ will contain a CMF that includes v_1.

To complete the proof of adequacy we need to show that condition C2 is satisfied, i.e., that every resulting simulation model $\mathcal{S}_s$ for a state s can be made complete by adding exogenous variables and that every such model determines the query variable v.

In building the scenario model we considered all the assumption classes that can affect v. The operating conditions of a CME from at least one of these assumption classes must be in $\mathcal{S}_s$, since otherwise that would imply that there is no model for the system. Therefore $\mathcal{S}_s$ includes the variable v. The library coherence assumption guarantees that the set of equations in $\mathcal{S}_s$, which we denote by Eq_s, is not over constrained. We need to show that there is a choice of exogenous variables which

does not include v that will make the equations Eq_s complete. If Eq_s includes m equations and $l > m$ variables, we need to choose $l - m$ exogenous variables. We can assume that Eq_s does not contain a complete subset of equations. If it does, there are two cases:

1. It contains a complete subset Eq_1 that includes v. In this case, we simply consider any choice of variables that makes $Eq_s - Eq_1$ complete. Since v is determined by some variable in Eq_1, the choice for $Eq_s - Eq_1$ will suffice.[10]

2. It contains a complete subset Eq_1 that does not include v. In this case, we solve the equations in Eq_1 and consider the equations Eq_2 resulting from replacing the appearances of variables from Eq_1 in $Eq_s - Eq_1$ by their solution values.[11] The resulting set of equations will not be over constrained (because we reduced the number of equations and the number of variables by the same number).

Let v_i be a variable in Eq_s. We show that set $Eq = Eq_s \cup Ex(v_i)$ is not over constrained.[12] Suppose, to the contrary, that it is over constrained. There would then be a subset of Eq that contains more equations than variables. That subset of Eq must contain the equation $Ex(v_i)$, because otherwise Eq_s would have been over constrained. Furthermore, that subset must contain v_i in some other equation as well. Now consider the set of equations $Eq - Ex(v_i)$. That set contains the same number of variables as in Eq, with one less equation. Therefore, it must either be over constrained, contradicting the assumption that Eq_s is not over constrained, or be complete, contradicting the assumption that Eq_s does not contain a complete subset of equations. Therefore, we can choose a variable in Eq_s which is not v and make it exogenous, and the set of equations will either be complete (in which case we are done) or will still be under constrained (in which case, we choose another variable). After choosing $l - m$ variables, the equations will be complete. Consequently, C2 holds.

Finally, we need to show that the model is as simple as possible. In this proof it is important to remember that we have only partial knowledge about the possible states that the system may reach. Specifically, all we know is that the time independent facts given in the system description must hold and that the value of binary variables given in E cannot change.

In the proof, we assume that $\mathcal{S}$ was constructed by adding the CMFs c_i from assumption class a_i at the ith iteration of **select-from-assumption-class**. Note that

[10]Note that if v belongs to a singleton set of complete equations, then this means that we have a model fragment that is modeling v as constant. Since the modeling conditions in the state are consistent, this is an adequate simulation model.

[11]Note that when solving a set of qualitative equations, the solution may contain some ambiguity (i.e., several possible solutions), we and need to consider each solution in turn.

[12]The equation $Ex(v_i)$ denotes that the variable v_i is exogenous.

some of the c_i's may have been removed subsequently by choosing a more complicated CMF from the same assumption class. We prove the following by induction on i:

A1. There must be a CMF in $\mathcal{S}$ from the assumption class a_i.

A2. The CMF c_i is the simplest CMF that can be chosen from a_i, w.r.t. $\mathcal{C}$.

A3. For each variable v_i on the queue, we must include all the phenomena that can affect v_i and can occur in one of the possible states of the system.

Conditions A1 and A3 guarantee that all the phenomena modeled in $\mathcal{S}$ are necessary. A2 guarantees that all these phenomena are modeled in the simplest way possible with respect to the modeling constraints, $\mathcal{C}$. The simplicity of $\mathcal{S}$ follows from these claims.

The base case includes all the assumption classes that can affect the query variable v. Clearly, A1 is satisfied because we need an assumption class that can determine v, and the ones that were chosen were those that are consistent with the possible states of the system. Since **select-from-assumption-class** selects the simplest model fragments in these assumption classes that do not contradict *Rel*, condition A2 is satisfied. Condition A3 is satisfied because if a variable v_1 appears with v in the same equation, then v_1 can causally influence v. If v_1 is not exogenous, then any phenomenon that can influence v_1 must be included in the model. Similarly, if v_1 appears in the operating conditions of a CMF that can determine v and is not exogenous, then any phenomenon that can affect v_1 must be included in the model.

We assume the claims for i and we prove them for $i + 1$. The CMF c_{i+1} could have been added in two ways. In the first, we use the outer loop (i.e., adding a new assumption class when popping a variable from the queue). By the inductive assumption, we must include all the phenomena that can affect the variable on the top of the queue. Therefore, adding CMF from a_{i+1} is necessary, and so A1 is satisfied. As before, A2 and A3 are satisfied because **select-from-assumption-class** selects the simplest CMF c that satisfies the assumptions made so far and adds only the necessary variables to the queue.

The second possibility for adding c_{i+1} is by the inner loop (i.e., by adjusting a previous choice from an assumption class). In this case, the inclusion of a CMF from a_{i+1} was justified by a previous CMF added to $\mathcal{S}$. Since the modeling assumptions in *Rel* include only those that are entailed by $\mathcal{C}$ and previous modeling assumptions, they are therefore the minimal set of assumptions, and since c_{i+1} is the simplest CMF from a_{i+1} that can be included in the scenario model, A2 is therefore satisfied.[13] Finally, the variables that were put on the queue when c_{i+1} is put in $\mathcal{S}$ are necessary using the same argument as before. Moreover, any variable that is already on the queue

[13]Note that if c_{i+1} was put in $\mathcal{S}$ instead of c_j for $j \leq i + 1$, then c_j was removed from $\mathcal{S}$

does not have to be removed, because the CMF c_{i+1} is replacing a CMF c_j that is a causal approximation of c_{i+1}, and therefore, any causal influence that was possible through c_j will be possible through c_{i+1}. ∎

The following theorem shows that $\mathcal{S}$ is built in time that is polynomial in the size of the problem:

Theorem 7.5: *Let d be the maximum number of CMFs in an assumption class and let n be the number of instantiated assumption classes in $\mathcal{S}$. Let l be the sum of the size of C and the number of ground atoms that appear in the instantiated modeling conditions of CMFs in $\mathcal{S}$. The running time of finding $\mathcal{S}$ is polynomial in n, d and l.*

Proof: Since the modeling constraints are Horn, computing the logical closure of the set of modeling assumptions is done in time polynomial in l. This is done every time we call the procedure **select-from-assumption-class**. The number of times this procedure is called is at most nd. This can be seen by observing that every call to **select-from-assumption-class** may, at worst, replace a CMF by another one that is more complicated than it. Since there are n instantiated assumption classes in $\mathcal{S}$ and at most d CMFs per class, this can only be done nd times. Consequently, the overall running time of the algorithm is polynomial in n, d and l. ∎

7.3.1 Relaxing the Assumptions

In this section we discuss the effect of relaxing some of the assumptions made in Theorem 7.4.

The Library Coherence Assumption

The most significant assumption that we made is the library coherence assumption. Although the assumption may seem a bit strong, there is a compelling argument for it. Specifically, if the assumption does not hold, this indicates a problem with the model fragment library. If we have a set of model fragments that satisfy the modeling constraints but give rise to an over constrained set of equations, this is an undesirable feature of the library, that calls for additional knowledge acquisition. It should be noted that the library coherence assumption is made implicitly in [Falkenhainer and Forbus, 1991]. In fact, if we assume (as in Qualitative Process Theory [Forbus, 1984]) that all equations are uniquely causally oriented, then the library coherence assumption follows when we make the causal approximations assumption and the assumption that the most complicated scenario model is adequate.

We can relax the library coherence assumption at the cost of doing more work at every state of the simulation. Specifically, in the absence of the coherence assumption, the scenario model created by our algorithm is guaranteed to produce a set of

equations at every state from which a complete model can be extracted (perhaps by removing some equations). We can extract the complete model efficiently using the methods described in [Nayak, 1992a].

Causal Approximations and Horn Restriction

The only role of the causal approximations assumption and the restriction that the modeling constraints must be Horn is to guarantee efficient performance of the model formulation algorithm. The causal approximations assumption guarantees that when we select a more complicated CMF in an assumption class, the set of simplifying modeling assumption decreases (i.e., more positive literals are added to *Rel*). The Horn restriction guarantees that once a positive literal has been put in *Rel*, it will not be retracted. Relaxing either of these two assumptions will require the algorithm to perform arbitrary backtracking and constraint satisfaction. As shown in [Nayak, 1992a], this will cause the model selection problem to be intractable.

7.4 Related Work

Several researchers have considered the problem of model formulation. Their work addresses one or both of the two aspects of the model formulation problem, namely model construction and model simplification.

Nayak [Nayak, 1992a] addressed both aspects. Nayak describes an algorithm for constructing a model for the single state case. His algorithm also follows possible causal influences; however, these influences must be given explicitly using the *component interaction heuristic*. In contrast, our work exploits the structure of the model fragments to derive these links, thereby not burdening the user with the error prone task of putting them in. It should be noted, however, that user intervention, as in Nayak's scheme, can enable a further focusing of the search by inserting only a subset of the links.

In choosing a model fragment from every assumption class, Nayak chooses the most complicated one, and then uses a procedure to simplify the resulting model. Our algorithm builds the model by selecting the simplest CMF possible in every class and only adjusts the choice if necessary. In cases where the CMFs in an assumption class vary significantly in their complexity, our approach leads to substantial savings in the search, since we only introduce the complicated models if necessary. It should be emphasized that the more complicated CMFs will involve more variables that will be put on the stack and will therefore result in a much larger scenario model. Finally, it should be noted that Nayak's methods for model simplification can be applied to the simulation model generated at every state from our scenario model.

Falkenhainer and Forbus' work on compositional modeling [Falkenhainer and Forbus, 1991] describes the representation aspects of compositional modeling and addresses the model construction problem. In their framework, every model fragment has a set of *relevance conditions* corresponding to our modeling conditions. Our use of relevance claims enriches their language (specifically their *Consider* predicate) and provides it with a formal basis. In their model formulation algorithm, they first select the physical scope of the model (by identifying the lowest object down the partonomic hierarchy that subsumes all the objects mentioned in the query) and then select the relevant properties of these objects. They rely on heuristics to select types of properties to be modeled. This approach can easily lead to inclusion of model fragments that are not causally related to the query, and it cannot guarantee the sufficiency of the model produced. Our algorithm provides more flexibility in that the selection of the physical scope of the scenario model and the selection of the relevant properties are done in a uniform way, (by reasoning about the modeling constraints) and can therefore affect each other. Furthermore, we only select properties to model that can causally influence the query. Finally, to select the simplest model, they generate all possible consistent sets of modeling assumptions and choose the simplest based on a very informal criteria of simplicity. Our selection of the simplest model is based on explicit representation of the differences between model fragments and on reasoning with formulas expressing these differences.

Rickel and Porter's work on model formulation [Rickel and Porter, 1992] is similar to ours since it makes use of graphs of interaction paths among variables to select relevant model fragments. Their graph of interactions is less general than the causal influence graph created by our algorithm, since it only includes variables, while we include all terms (including variables, predicates and relations) that could directly or indirectly influence the goal terms. Their approach also does not provide guarantees of sufficiency or simplicity.

The idea of explicitly representing the differences between CMFs in an assumption class is similar to the graph of models by Addanki et al. [Addanki *et al.*, 1989]. Their work addresses the problem of selecting among complete models. Since the models in their graph are complete models instead of fragments, the space requirement of their approach increases exponentially as the number of possible modeling assumptions increases. Our approach can be viewed as combining the idea of a graph of models with compositional modeling.

The model simplification problem has been addressed by Williams [Williams, 1990a] and Weld [Weld, 1990]. Williams also makes use of causal influence graphs to simplify a model. Both Weld and Williams assume a complete model of the situation as an input. Williams also makes use of the idea of following causal influences in his work on innovative design [Williams, 1990b].

7.5 Summary and Contributions

This chapter described an application of relevance reasoning to the domain of modeling physical systems. Aside from showing that relevance reasoning is a viable approach to solving the model formulation problem, we have also shown that the modeling problem can significantly benefit from being considered from the perspective of relevance reasoning. Specifically, we have shown that some aspects of the modeling problem can be approached using general considerations of relevance reasoning. (i.e., backward chaining on causal influences and articulating the differences between CMFs in assumption classes). Moreover, we have shown how to incorporate engineering knowledge and heuristics for modeling in a declarative fashion, using relevance reasoning. The ability to declaratively express modeling heuristics has several advantages. Since it is easier to inspect and modify declarative knowledge, experimenting with different modeling heuristics becomes viable. In contrast, other methods wire in their modeling heuristics, and therefore modifying them requires rewriting code. The result of our approach was a novel model formulation algorithm which efficiently selects the simplest model for a system and a query. An important aspect of our algorithm is that it chooses a model for a simulation of the system without knowing precisely which states the system can reach.

The algorithm has been implemented as part of a system called Device Modeling Environment (DME) [Iwasaki and Low, 1992], which is a device modeling program to provide a computational environment for design of electromechanical devices. Given a topological description of a device, DME formulates a behavior model of the device using the compositional modeling approach and simulates its behavior. Prior to implementing our algorithm, the system would prompt the user to select a set of model fragments to be considered in the scenario model, thus creating a significant knowledge acquisition bottleneck. DME checks the operating conditions of every model fragment in the scenario model to determine the simulation model for each state. The system works on several examples, including the electrical power system of an earth orbiting satellite, of which the example used in this chapter is a much simplified version.

Research on compositional modeling is in its infancy. The discussion in this chapter contributes by crystalizing some of the main questions regarding the approach that require additional research. The key issues that came to bear in this chapter are (1) how to write model fragments (i.e., how to decide what phenomena can be described in a single model fragment, and what assumptions to make regarding the contents of a model fragment), (2) how to organize model fragments in a library and (3) what assumptions can be made about the model fragment library. We have contributed to solving problem (2) by suggesting the concept of compositional model fragments and

by requiring explicit representation of the differences between CMFs in an assumption class. In our discussion we made several assumptions regarding questions (1) and (3). In general, we see a tradeoff between (1) and (3). If more assumptions are made about individual model fragments, then fewer constraints need to be placed on the model library as a whole, and vice versa. Finding the optimal point in the spectrum of possibilities requires additional research and practical experience building systems. We believe that imposing some structure on the model fragment library is necessary and beneficial in the long run to facilitate knowledge acquisition and reuse.

Finally, as mentioned in Chapter 6, the problem of model formulation can be viewed as one instance of a problem solving setting in which a system needs to reason *about* its own knowledge before answering a query. In doing so, it must choose among alternative representations of the domain that make different assumptions and abstractions. Other instances of this problem are also currently under investigation, such as reasoning with contexts and query evaluation in heterogeneous databases. We believe that the techniques developed in this chapter can form the basis for reasoning mechanisms in these other problem solving tasks.

Chapter 8

Conclusions

The ability to automatically identify and ignore irrelevant information is a key to providing efficient inferences from large knowledge based systems and for a system to be able to create appropriate abstractions in a complex domain. The main contribution of this dissertation is showing that it is possible to reason effectively about relevance of knowledge in a principled manner and that such reasoning can significantly impact the preformance of knowledge based systems. This chapter begins by summarizing the specific contributions of this dissertation. We then present a tabular summary of the main references to related work. Detailed discussion of related work is scattered at the relevant points throughout the dissertation. Finally, section 8.2 concludes with a description of directions for future research.

8.1 Summary of Contributions

The two key issues that need to be addressed in relevance reasoning are (1) how to automatically decide what knowledge is irrelevant to a query and (2) what is the utility of relevance reasoning. As a basis for addressing these two issues, we presented a formal framework for analyzing irrelevance. The framework included a space of possible definitions of irrelevance based on a proof theoretic analysis of the notion. The framework enabled us to compare the properties of different irrelevance claims. Within the space of definitions, we identified the class of strong irrelevance claims that has two desirable properties; namely, strong irrelevance claims can be efficiently derived automatically and are guaranteed to lead to savings in inference. The framework also shed new light on the problem of deciding when a query is independent of an update to the knowledge base and enabled us to significantly extend previous results in this area.

The framework provided a setting in which we could investigate the connection between the notion of irrelevance and the creation of abstractions. This connection led

Title and reference	Page(s)
Analysis of (ir)relevance:	
A theory of irrelevance [Subramanian and Genesereth, 1987; Subramanian, 1989]	39, 101, 144
In the philosophy literature [Keynes, 1921; Carnap, 1950; Gärdenfors, 1978]	39
Relevance logics [Anderson and Belnap, 1975; Avron, 1992]	39
In probabilistic reasoning [Pearl, 1988]	39
Static Analysis of rules/clauses:	
Connection graphs [Kowalski, 1975; Sickel, 1976; Chang, 1979]	76
Static analysis in Explanation Based Learning [Etzioni, 1993; Etzioni, 1990]	75, 97
Pushing constraint selections [Srivastava and Ramakrishnan, 1992]	75
Partial evaluation of logic programs [Smith and Hickey, 1990; Lloyd and Shepherdson, 1991; Bruynooghe *et al.*, 1991]	75, 100
Tree automata [Vardi, 1989]	74
Automated reasoning and query evaluation:	
Knowledge compilation [Selman and Kautz, 1991]	100
Deriving optimal search strategies [Smith, 1986; Greiner, 1991]	97
Message passing based query evaluation [Van-Gelder, 1986]	96
Magic set transformation [Ullman, 1989; Mumick *et al.*, 1990]	94
Independence of queries from updates:	
Detecting independence [Blakeley *et al.*, 1989; Elkan, 1990]	112, 126
Conjunctive query containment [Klug, 1988; van der Meyden, 1992]	126
Abstractions:	
Predicate abstraction [Plaisted, 1981; Tenenberg, 1990]	144
Projecting existential arguments [Ramakrishnan *et al.*, 1988]	144
A theory of abstraction [Giunchiglia and Walsh, 1992]	145
Automatic creation of abstractions [Knoblock, 1990; Knoblock *et al.*, 1991]	146
Modeling physical devices:	
Compositional modeling [Falkenhainer and Forbus, 1991]	170
Causal approximations [Nayak, 1992a]	169
Graphs of models [Addanki *et al.*, 1989]	170
Model simplification [Weld, 1990; Williams, 1990a]	170

Table 8.1: References to related work.

to a new approach to research on reasoning with abstractions in which we investigate the properties of an abstraction by considering the irrelevance claims on which it is based. We demonstrated the approach on the cases of predicate abstraction and argument projection. In both cases, the analysis of the corresponding irrelevance claims led to efficient algorithms for automatically creating abstractions and to better understanding of the utility of the resulting abstractions.

We investigated in detail the problem of automatically deriving irrelevance claims for Horn rule knowledge bases and several extensions. Our analysis was based on the observation that in order for relevance reasoning to be practical, we must derive irrelevance claims by considering only a small and stable part of the knowledge base, while not assuming anything about the unexamined parts. We considered the problem

of automatically deriving irrelevance claims that were based only on the rules in the KB and were independent of the ground formulas. As a result, our algorithms were efficient and the irrelevance claims derived were independent of changes to the ground formulas.

Our algorithms for deriving irrelevance were based on a novel tool, the query-tree, which is one of the main contributions of this work. The query-tree is a finite structure that gives us a *view* of the knowledge base. It encodes precisely the set of possible derivations of the query. Consequently, it tells us exactly which rules and ground formulas can appear in derivations of the query, thus providing the basis for a sound and complete inference procedure for several classes of strong irrelevance claims. One of the key aspects of the query-tree is that it considers the semantics of the interpreted literals that appear in the rules, which often enables us to detect additional interactions between the rules. The query-tree can also be built to encode only the minimal derivations of the query, or only the satisfiable derivations in cases where EDB literals may appear negated in the antecedents of the rules. We also showed how the query-tree can be used to derive logical consequences of irrelevance claims that are given to the system by an external source, and to guide the search of a backward chainer so that it follows only paths that can yield derivations of the query.

We presented experimental results which showed that using the query-tree to filter out irrelevant formulas often yields speedups of orders of magnitude, while the cost of building the query-tree is negligible. Additional speedups were obtained by using the query-tree to guide the search of the backward chainer. Both the theoretical analysis and the experimental results showed that our methods will scale up and be even more effective in larger knowledge bases.

Finally, we applied the relevance reasoning framework to the domain of modeling physical devices. We considered the task of selecting a model for a device to answer a given query by composing model-fragments, each describing a single phenomenon in the physical world at different levels of abstraction and using different approximations. We presented a novel model selection algorithm based on relevance reasoning. The algorithm used relevance reasoning in order to (1) determine which phenomena are relevant to the query (and therefore should be included in the model), and (2) to reason about the abstractions underlying the model-fragments which present multiple descriptions of these phenomena in order to determine the abstraction level most appropriate for the query.

8.2 Future Work

The work described in this dissertation can be extended in several ways. In this section we describe several directions for future research.

Analysis of Irrelevance

Our framework for analyzing irrelevance should be extended in several ways in order to make it applicable in a wider variety of settings. One important extension is to incorporate probabilistic irrelevance claims into the framework, i.e., claims stating that some formula in the knowledge base is irrelevant to a query with some probability. A clear understanding of the meaning of such irrelevance c' .ns is needed as well as algorithms for automatically deriving them and methods for exploiting them in inference. Second, our analysis focused on the case in which answering a query is done by searching for a derivation, using some given set of inference rules. Although many problem solving situations can be cast in that way, doing so will not always yield useful results. Therefore, an important extension to the framework is to formalize irrelevance for general problem solving. For example, whereas now the framework revolves around the possible derivations of a query, a more general framework will revolve around paths in a state space. Finally, in our discussion we considered only the cases in which reasoning is monotonic. An interesting problem is to define irrelevance (and develop the corresponding algorithms) in the setting of non-monotonic reasoning.

As mentioned in Chapter 2, the notion of irrelevance is very closely related to the notion of belief revision. A more thorough investigation of this connection could yield interesting results. On the one hand, considering definitions of irrelevance based on belief revision will yield more semantically based definitions of irrelevance. On the other hand, associating a definition of belief revision with irrelevance may shed light on the plethora of definitions of belief revision. Additionally, the problem of developing efficient algorithms for belief revision has received little attention to date. Algorithms for deriving irrelevance claims might prove to be a key tool in developing efficient belief revision algorithms.

The Query-tree

The query-tree has proven to be a powerful tool in relevance reasoning and controlling inference, and it is therefore interesting to extend it to a wider class of languages. One important extension is to widen the class of interpreted constraints that can be handled by the query-tree. Currently, except for constraint literals in the rules, the query-tree can also fully incorporate constraints that are given on the arguments of a relation in the knowledge base. One way to extend the tree is to consider constraints that include arguments from more than one relation (a.k.a. integrity constraints). An example of such a constraint is stating that a join of two (or more) relations is empty. Another important extension to the query-tree is to consider rules with function symbols. Although in general, the query-tree cannot provide a complete inference procedure for strong irrelevance when function symbols are present, it is important to find limited cases in which such a procedure can be found. In cases

where completeness cannot be guaranteed, one could develop methods that will detect a wide class of irrelevance claims encountered in practice.

The use of the query-tree to control inference should also be investigated further. The uses we described and with which we experimented were straightforward applications of the query-tree. As discussed in Chapter 4, the query-tree also enables generalization other query-optimization methods such as Magic-sets and message passing schemes. Given the query-tree, we are in a position to devise a more general framework for query-optimization that will incorporate Magic-sets, message passing, tail recursion optimizations and pushing selections and projections.

Finally, a key contribution of the query-tree is that it provides descriptions of specialized indices for accessing the ground formulas in the knowledge base. These indices are tailored for a specific set of queries. An important question that needs to be addressed in the context of any large system, is how to combine the indices given by the query-tree with current database indexing techniques.

Irrelevance and Abstractions

One of the major areas for future work spawned by this dissertation is the connection between the notion of irrelevance and the creation of abstractions, as described in Chapter 6. The approach proposed there is to associate an abstraction with an irrelevance claim, stating which knowledge is removed in the abstract theory. An understanding of the abstraction is obtained by an analysis of the corresponding irrelevance claim. Chapter 6 listed several kinds of irrelevance claims that should each be investigated further. Of particular interest are the questions of (1) finding algorithms for automatically creating an appropriate abstraction, (2) understanding the utility of reasoning with the abstraction and (3) determining when and how abstractions can be composed (by composing the corresponding irrelevance claims). As we saw in Chapter 6, our treatments of irrelevance of predicate arguments and predicate refinements had many similarities. A longer term goal is to develop a general framework for treating a large class of irrelevance subjects.

Modeling of Physical Systems

Compositional modeling is a powerful paradigm for building systems that reason about physical systems. However, the basic building blocks of the approach require a much better understanding. Specifically, we need a clearer definition of what is a model-fragment, i.e., what phenomena should be considered a single model fragment, and what assumptions can we make about model fragments. Second, we need to understand how to build libraries of model fragments. Such libraries are not random collections of model-fragments, but rather have a lot of implicit and explicit structure. Significant leverage in devising compositional modeling algorithms will come

from discovering the underlying structure and exploiting it. The work described in Chapter 7 makes a few contributions in this direction, but much work remains to be done. Building large experimental systems will provide significant insights into these issues.

Finally, we cannot expect a model-fragment library to contain model-fragments that describe a certain phenomenon at all possible levels of abstraction that may be needed in solving queries. Therefore, an interesting research problem is to automatically create model-fragments with the desired level of abstraction by abstracting model-fragments from the library.

8.3 Final Word

The work described in this dissertation is at the border between artificial intelligence and database systems. I believe that research combining techniques from these two fields will be of prime importance in future years. One of the major technological innovations in upcoming years will be the availability of large amounts of information in practically every household. Developing systems that will provide intelligent access to information presents a unique opportunity in which techniques from both artificial intelligence and databases will make a great impact on society. I believe that the combination of techniques from these two fields, as demonstrated in this dissertation, will provide the essential building blocks of successful systems of the information age.

Appendix A

Proofs

A.1 Proofs of Chapter 3

Proof of Theorem 3.4

Proof: In the proof it is more convenient to refer to the relations denoted by the labels, rather than the labels themselves. The conjunction of two labels c_1 and c_2 is represented by the *join* of their corresponding relations, denoted by $R_{c_1} \bowtie R_{c_2}$. Recall that since the constraint language satisfies the Closure property, we can express a join of two relations, and a projection of a relation on a subset of its variables as a sentence in the given constraint language $\mathcal{L}$. We denote the relations represented by $c_0(n), c_b(n)$ and $c_f(n)$ by $R_0(n), R_b(n)$ and $R_f(n)$ respectively. We denote the projection of a relation R on a subset of its variables $\bar{X}$ by $R|_{\bar{X}}$.

Let $r_1, \ldots, r_l$ be the top-down ordering of the rule-nodes in d that was used in the second phase of the algorithm. Recall that by the definition of c_d (the global constraint on the variables in d), $c_d = c_0(r_1) \wedge \ldots \wedge c_0(r_l)$, or in notation of relations, $R_d = R_{c_0(r_1)} \bowtie \ldots \bowtie R_{c_0(r_l)}$. We define a sequence of relations as follows:

- $R_0 = R_b(root(d))$
- $R_i = R_{i-1} \bowtie R_b(r_i)$

We prove that the following properties hold for the sequence $R_0, \ldots, R_l$:

D1: $R_l = R_d$, i.e., the final relation is the same as the global constraint on d.

D2: If $\bar{X}_i$ is the set of variables that appear in R_i, then

D2.1: $R_{i+1}|_{\bar{X}_i} = R_i$ and

D2.2: $R_i|_{r_i} = R_f(r_i)$.

This means that once a set of variables appears in an intermediate relation, the subsequent relations do not change with respect to that subset.

The theorem follows from properties $D1$ and $D2$ as follows. Let r_i be the i^{th} rule-node in d and let X_i be the variables that appear in the father or the children of r_i. It follows from $D2$ that $R_f(r_i) = R_l|_{X_i}$, and by $D1$, that $R_l = R_d$. Therefore, $R_f(r_i) = R_d|_{X_i}$ holds which is the way $R_f(r_i)$ is defined.

To prove $D1$, we observe that $R_l = R_b(r_1) \bowtie \ldots \bowtie R_b(r_l)$ (i.e. the join of the labels obtained in the bottom-up phase). Therefore, it is enough to show that $R_d = R_b(r_1) \bowtie \ldots \bowtie R_b(r_l)$. To show this we prove that for every rule node r in the tree and its father node g the following hold:

(a) $R_b(r) \subseteq R_0(r)$, $R_b(g) \subseteq R_0(g)$, and

(b) $R_b(r) \supseteq R_d|_r$, $R_b(g) \supseteq R_d|_g$

Since $R_d = R_0(r_1) \bowtie \ldots \bowtie R_0(r_l)$, (a) gives us that $R_l \subseteq R_d$. From the properties of the join operation and from (b) we get $R_l \supseteq R_d$. Hence, $R_l = R_d$.

The proof of (a) and (b) proceeds by induction on the elements of $r_1, \ldots, r_l$ in reverse order (i.e., the bottom-up order). Note that the second parts of (a) and (b) follow from their first parts, since $R(g)$ is the projection of $R(r)$ on the variables in g. The base case consists of all the rule nodes whose children are all leaves. For each such node r, $R_b(r) = R_0(r)$ and therefore, (a) and (b) hold trivially.

Assume (a) and (b) hold for all rule nodes $r_{i+1}, \ldots, r_l$. We need to show that it holds for r_i. Claim (a) holds because $R_b(r_i)$ is the intersection of $R_0(r_i)$ with the bottom-up labels of its children. To prove (b), let $g_1, \ldots, g_m$ be the children of r_i. By the induction assumption, $R_b(g_i) \supseteq R_d|_{g_i}$ for each subgoal g_i. Clearly, by the definition of $R_d|_{r_i}$, $R_0(r_i) \supseteq R_d|_{r_i}$. Therefore, since $R_b(r_i)$ is actually the join of relations that all contain $R_d|_{r_i}$,[1] $R_b(r_i) \supseteq R_d|_{r_i}$.

We prove $D2$ by induction on i. For the base case $i = 0$, we note that R_1 is simply $R_b(r_1)$. Therefore, since the $R_b(root(d))$ is the projection of $R_b(r_1)$ on the variables of $root(d)$, D2.1 holds. Moreover, since $R_f(r_1) = R_b(r_1)$, D2.2 holds too.

We assume the claim holds for all $j < i$, and prove that it holds for i. We first prove D2.2. Let g be the father of r_{i+1} and r_i be the father of $\hat{g}$. By the inductive assumption, $R_i|_{r_i} = R_f(r_i)$, and therefore the same holds for the goal node $\hat{g}$, i.e., $R_i|_{\hat{g}} = R_f(\hat{g})$. Moreover,

$$R_f(r_{i+1}) = R_f(g) \bowtie R_b(r_{i+1}) = R_i|_g \bowtie R_b(r_{i+1}).$$

[1] More precisely, the relations $R_b(g_1), \ldots, R_b(g_m), R_0(r_i)$ contain the respective projections of R_d on their variables.

However, since the variables common to r_{i+1} and to R_i are only those in g, the join and the projection commute, i.e.,

$$R_f(r_{i+1}) = (R_i \bowtie R_b(r_{i+1}))|_{r_{i+1}}.$$

To show D2.1, it is enough to show that $R_i|_g = R_{i+1}|_g$, since the variables appearing in g are the only ones common to R_i and $R_b(r_{i+1})$, or equivalently, we can show

$$R_f(r_{i+1})|_g = R_f(g). \tag{A.1}$$

The proof uses the following observation:

If A is the projection of a relation R on a subset of its variables, and $B \subset A$, then joining B with R and projecting on the same variables will result in the relation B.

In our case,

$$R_b(r_{i+1})|_g = R_b(g). \tag{A.2}$$

Clearly, $R_f(r) \subseteq R_b(r)$ and $R_b(r)|_g \subseteq R_b(g)$, and therefore, since $R_f(g) = R_f(r)|_g$ it follows that

$$R_f(g) \subseteq R_b(g). \tag{A.3}$$

Finally, recall that

$$R_f(r_{i+1}) = R_f(g) \bowtie R_b(r_{i+1}). \tag{A.4}$$

Therefore, the above observation together with A.2, A.3 and A.4 entail A.1. ∎

Proof of Theorem 3.10.

We begin by defining an intermediate language, $\mathcal{L}_s$, which is less expressive than $\mathcal{L}^{\wedge,\vee}$ but more expressive than $\mathcal{L}^{\wedge}$, and will have the Closure property. We denote by $\mathcal{L}^{\wedge}_{\leq}$ the language that allows only the predicate $\leq$ and conjunction, and by $\mathcal{L}^{\wedge,\vee}_{\neq}$ the language that allows both disjunction and conjunction but only the predicate $\neq$. Note that all the predicates can be expressed by conjunctions of $\leq$ and $\neq$.

Definition A.1: The language $\mathcal{L}_s$ contains all the sentences of the form $\phi \wedge \psi$, where $\phi \in \mathcal{L}^{\wedge}_{\leq}$ and $\psi \in \mathcal{L}^{\wedge,\vee}_{\neq}$. ∎

Lemma A.2: *The language $\mathcal{L}_s$ has the Closure property.*

Proof: The join of two relations represented by the two sentences $\phi_1 \wedge \psi_1$ and $\phi_2 \wedge \psi_2$ is simply the sentence $(\phi_1 \wedge \phi_2) \wedge (\psi_1 \wedge \psi_2)$, which is in $\mathcal{L}_s$. Selection can be expressed by simply adding conjuncts of the form $X_i = X_j$ or $X_i = c$. To show that $\mathcal{L}_s$ is closed under projection, let $c = \phi \wedge \psi$ we a sentence in $\mathcal{L}_s$. We can assume that ψ is in disjunctive normal form. Let $\bar{X}$ be a subset of the variables in c.

The language $\mathcal{L}^{\wedge}_{\leq}$ is has the closure property (it simply represents a transitive relation on its variables). Therefore we have a sentence $\phi_{\bar{X}}$ describing the exact projection of ϕ on $\bar{X}$.

Given a tuple $\bar{a}$ that satisfies $\phi_{\bar{X}}$, it will be in the projection of c on $\bar{X}$ if it can be extended to the variables in c in such a way that ψ is satisfied. However, we note that a contradiction between an extension of $\bar{a}$ and ψ can only arise from that fact that the extension satisfies too many equalities.

Therefore, in order to construct a sentence that is equivalent to $c|_{\bar{X}}$, all we need to do is check all the possibilities for equalities between the variables of c, and exclude the ones that contradict ψ. Specifically, let k be a partition of the variables $\bar{X}$ and the constants appearing in c, and let $c_{=}(k)$ be the conjunction of all the atoms $X = Y$, where X and Y are in the same partition. Let $k_1, \ldots, k_n$ be the partitions for which $c \wedge c_{=}(k)$ is unsatisfiable. We define $c_{\bar{X}}$ by

$$c_{\bar{X}} = \phi_{\bar{X}} \wedge \neg c_{=}(k_1) \wedge \ldots \wedge \neg c_{=}(k_n)$$

It is easy to see that any tuple not satisfying this formula will not be a member of $c_{\bar{X}}$ (since it either violates $\phi_{\bar{X}}$, or belongs to one of the unsatisfiable partitions). Any tuple that does satisfy this formula satisfies $\phi_{\bar{X}}$, and furthermore the equalities that it satisfies are consistent with c. ∎

Suppose d is a symbolic derivation tree and we are computing the labeling L_{sat}. Recall that in the rules we can only have conjunctive constraints (i.e., the conjunction of the literals of interpreted predicates). These constraints are therefore expressible in $\mathcal{L}_s$. Furthermore, labels L_{sat} are computed by join and projection operations on these constraints, and therefore, since $\mathcal{L}_s$ has the Closure property, all the labels in L_{sat} can be expressed in $\mathcal{L}_s$. This observation enables us to characterize the difference between the labels created by $\mathcal{L}^{\wedge}$ and $\mathcal{L}^{\wedge,\vee}$:

Lemma A.3: *Let d be a symbolic derivation tree and let $n \in d$. Let $c_b(n) = \phi \wedge \psi$, be the bottom-up label of n, where $\phi \in \mathcal{L}^{\wedge}_{\leq}$ and $\psi \in \mathcal{L}^{\wedge,\vee}_{\neq}$; then $c^{\wedge}_b(n) = \phi \wedge \psi_1$, where $\psi \models \psi_1$. The same relationship holds between $c^{\wedge}_f(n)$ and $c_f(n)$.*

Proof: The key observation underlying the proof is the following. Suppose $c = \phi \wedge \psi$ is a sentence in $\mathcal{L}_s$, and $\bar{X}$ is a subset of its variables. Let $c_{\bar{X}} = \phi_1 \wedge \psi_1$ be the projection of c on $\bar{X}$. Suppose that $\tilde{r}$ is the sentence in $\mathcal{L}^{\wedge}$ that is most close to (still

weaker than) c_X (note that "most close" is well defined and unique). If τ is written in $\mathcal{L}_s$ as $\phi_2 \wedge \psi_2$, then $\phi_1 = \phi_2$ and $\psi_1 \models \psi_2$.

Therefore, by projecting a sentence into $\mathcal{L}^\wedge$ we only make the $\neq$ part weaker. Based on this observation, we can prove the lemma by a bottom-up induction on the nodes of d followed by a top-down induction.

For the leaf nodes, the claim is trivially true, since $c_b(n) = c_b^\wedge(n)$. Consider a rule-node r, and suppose the claim holds for all subgoals of r, $g_1, \ldots, g_m$. The label $c^\wedge(r)$ is computed by conjoining $c(r)$ and $c^\wedge(g_1), \ldots, c^\wedge(g_m)$. Since the conjunction can be done separately for the two components of the labels and since the labels of $g_1, \ldots, g_m$ satisfy the inductive assumption, the claim will also hold for $c_b(r)$. To compute the label of the father g of r, we project $c^\wedge(r)$ on the variables of g. Based on the observation above, and since the claim holds for r, the resulting label of g can only be weaker than $c_b(g)$ in the second ($\neq$) component. The proof is completed by a top-down induction on the nodes of d in a similar fashion. ∎

Proof of Theorem 3.10: Part 1 of the theorem follows directly from Lemma A.3. Part 2 can be shown as follows. Suppose that $c_f(n) = \phi \wedge \psi$ and that $c_f^\wedge(n) = \phi \wedge \psi_1$ such that $\psi \models \psi_1$. We can assume that ψ is in disjunctive normal form and at least one of its disjuncts (assume it is the first) is satisfiable in conjunction with ϕ. Suppose its first disjunct u_1 is $\neg p_1 \wedge \ldots \wedge \neg p_l$, where each of the p_i's is an equality. Since $\phi \wedge u_1$ is satisfiable, that means that $\phi \not\models p_i$ for any $1 \leq i \leq l$. Therefore, $\neg p_i$ will be a conjunct in $Max_{\neq}(c_f^\wedge(n))$ (note that $Max_{\neq}(c_f^\wedge(n))$ can be computed using ϕ alone). Consequently $Max_{\neq}(c_f^\wedge(n)) \models u_1$ and therefore $Max_{\neq}(c_f^\wedge(n)) \models \phi \wedge \psi$.

Finally, the third part of Theorem 3.10 is proved as follows. Starting from the root of d, we show that if all the labels of $c_f^\wedge(n)$ are satisfiable, then we can construct a mapping ψ of the variables of d to constants that satisfies all the constraints in the rules. Therefore, if we have such a variable mapping, it must be the case that c_d is satisfiable and therefore $c_f(n)$ is satisfiable for every $n \in d$.

We begin by assigning values to the variables that appear in the root of d, in a way that is consistent with $c_f(root(d))$. We assign distinct values to variables X_i and X_j unless $c_f(root(d))$ implies that $X_i = X_j$ (and unless $X_i = a$ is implied, we assign to X_i a value other than a).

We construct the variable mapping ψ in a top-down order on the rule-nodes in d. Let r be a rule-node with father g and interpreted subgoals c_r. We assume that the variables of g have already been assigned values that are consistent with $c_f^\wedge(g)$ and contain only equalities that are implied by $c_f^\wedge(g)$. We extend ψ to the variables that appear in subgoals of r but do not appear in g. In doing so, we choose values that are consistent with $c = c_f^\wedge(g) \wedge c_r$, but only contain equalities that are implied by c. To complete the proof, we need to show:

1. The constraint c is satisfiable

2. $(g\psi) \in c|_g$.

The first condition guarantees that c can be satisfied, and the second guarantees that it can be satisfied by extending the mapping created thus far (for the variables of g). To prove 1, recall that $c_f^\wedge(r)$ is satisfiable. However, $c_f^\wedge(r)$ was computed by first computing $c_1 = c_f^\wedge(g) \wedge c_b^\wedge(r)$, and then finding the strongest constraint in $\mathcal{L}^\wedge$ that is weaker than c_1. Therefore c_1 must have also been satisfiable. Moreover, $c_b^\wedge(r) \models c_r$, and therefore, c is satisfiable.

To prove 2, suppose $c_f^\wedge(g) = \phi_1 \wedge \psi_1$ when written in $\mathcal{L}_s$, and suppose $c = \phi_2 \wedge \psi_2$. Recall that $\phi_2|_g = \phi_1$ because the language $\mathcal{L}_\leq^\wedge$ has the Closure property. Moreover, $g\psi$ satisfies $c_f^\wedge(g)$ and has only equalities that are implied by $c_f^\wedge(g)$. Therefore, $g\psi$ will satisfy ψ_2 and will therefore satisfy $c|_g$. ∎

Proof of Theorem 3.26

We begin by considering the case of $SI(r, q, \Sigma_\mathcal{P}, DI_2, \mathcal{D}_q)$ when the rules have the predicate $\neq$. The theorem is proved by reducing the acceptance problem of a linear-space alternating Turing machine (ATM) [Chandra *et al.*, 1981] to the problem of finding irrelevance of rules. The execution of an ATM is described by a sequence of instantaneous descriptions, id's, each describing the state of the machine at consecutive stages of the execution, i.e., the contents of the input tape, the location of the head and the state of the machine. An ATM is similar to a Turing machine, except that its transition function gives a pair of moves for each combination of state and symbol. Furthermore, every state is either an *and-state* or an *or-state*. If q is an and-state, then an id having state q leads to acceptance of the input if both its successors lead to acceptance. If q is an or-state, then an id having state q leads to acceptance if either one of its successors leads to acceptance. The states of the machine alternate in the sense that the successors of an and-state are or-states, and the successors of an or-state are and-states.

Instantaneous descriptions are represented by a symbol for every cell on the tape. The symbol can either be an input symbol, or a composite symbol including an input symbol and a state of the machine. In a legal id, all cells but one contain the input symbol that is on the tape in that state, and the cell on which the head is placed contains a composite symbol containing the input symbol in that cell and the internal state of the machine. The union of the input symbols and composite symbols is denoted by B.

The reduction is based on representing id's as tuples of a predicate *id*, whose arity is linear in the size of the input tape, n. Each cell in the id is represented by a block of variables of size $|B|$. The variable X appears in the block in the position corresponding to the symbol appearing in the cell (we assume some arbitrary ordering

on the elements of B). All other columns contain the variable W. Thus the arity of the predicate id is $|B|n$. The tuples $\vec{X}_i$ are used to denote blocks of variables corresponding to one cell. The tuple $\vec{U}_i$ denotes a block of variables representing a cell with the symbol i, (i.e., X appears in the position of i in B and all other positions are W).

Intuitively, we construct the program such that $id(\vec{X})$ is derivable if and only if $\vec{X}$ describes a legal id and leads to acceptance. Given an ATM, M, and an input X_{init}, we construct a program as follows. First we need rules representing transitions between consecutive states. Suppose $\delta(c,q) = \{(d_1, s_1, R), (d_2, s_2, L)\}$[2] is a transition of M. If q is an or-state, then for every i, $(1 \leq i \leq n)$[3] and every input symbol b, the program contains the following rules:[4]

$$id(\vec{X}_1, \ldots, \vec{X}_{i-1}, \vec{U}_{d_1}, \vec{U}_{(s_1,b)}, \vec{X}_{i+2}, \ldots, \vec{X}_n) \Rightarrow id(\vec{X}_1, \ldots, \vec{X}_{i-1}, \vec{U}_{(c,q)}, \vec{U}_b, \vec{X}_{i+2}, \ldots, \vec{X}_n)$$

$$id(\vec{X}_1, \ldots, \vec{X}_{i-2}, \vec{U}_{(s_2,b)}, \vec{U}_{d_2}, \vec{X}_{i+1}, \ldots, \vec{X}_n) \Rightarrow id(\vec{X}_1, \ldots, \vec{X}_{i-2}, \vec{U}_b, \vec{U}_{(c,q)}, \vec{X}_{i+1}, \ldots, \vec{X}_n)$$

If q is an and-state then for every i, $(1 \leq i \leq n)$ and every pair of input symbols b_1, b_2, the program contains the following rule:

$$\begin{aligned} &id(\vec{X}_1, \ldots, \vec{X}_{i-2}, \vec{U}_{b_1}, \vec{U}_{d_1}, \vec{U}_{(s_1,b_2)}, \vec{X}_{i+2}, \ldots, \vec{X}_n) \wedge \\ &id(\vec{X}_1, \ldots, \vec{X}_{i-2}, \vec{U}_{(s_2,b_1)}, \vec{U}_{d_2}, \vec{U}_{b_2}, \vec{X}_{i+2}, \ldots, \vec{X}_n) \Rightarrow \\ &\quad id(\vec{X}_1, \ldots, \vec{X}_{i-2}, \vec{U}_{b_1}, \vec{U}_{(c,q)}, \vec{U}_{b_2}, \vec{X}_{i+2}, \ldots, \vec{X}_n) \end{aligned}$$

To complete the program, a few more rules are necessary. Denote by $\vec{X}_{final}$ the tuple representing M's (unique) accepting state, and by $\vec{X}_{init}$ the tuple representing the initial state. R1 places the initial state as a subgoal of the query:

R1: $id(\vec{X}_{init}) \Rightarrow p(X, W)$

R2 and R3 will lead from the accepting state to an EDB node. Note that e is the only EDB predicate.

R2: $a(X, W) \Rightarrow id(\vec{X}_{final})$
R3: $(X \neq W) \wedge e(X, W) \Rightarrow a(X, W)$

We denote the set of rules by $\mathcal{P}$. The following theorem establishes the correctness of the reductions.

[2]The L and R are arbitrary.

[3]If $i = n$, the head cannot move to the right, if $i = 1$, it cannot move to the left.

[4]The exact form of the rule depends on the direction of the head movement. These rules are shown to reflect the transition shown.

Theorem A.4: *$SI(\mathtt{R1}, p, \Sigma_{\mathcal{P}}, DI_2, \mathcal{D}_{\hat{p}})$ holds if and only if M does not accept its input with the initial state X_{init}.*

Proof: We first note that any derivation of $p(X, W)$ will contain only two constants. This follows from the fact that in all rules, all variables appearing in the body of the rule appear in the head too. Therefore, the only constants in the derivation will be those assigned to X and W. Furthermore, since a derivation must include the rule R3, these constants must be distinct. Therefore, we will refer to them as X and W hereafter. Moreover, if X and W are distinct, an *id* subgoal can only be unified with itself in the head of a rule. This can be seen by considering each block in the *id*. If they differ in the position of the X variable (or if one of them does not contain exactly one occurrence of X), it will force X and W to unify[5]. Therefore, because of the way the transition rules are written, the subgoals of any *id* node are the instantaneous descriptions of its successor states. Consequently, if the top *id* goal-node in a derivation is the node describing the initial state, then every partial derivation of p describes a possible execution tree of the ATM M. Therefore, because the only way to get to and EDB subgoal is through rules R2 and R3, every derivation of p must describe an accepting execution trace of M. Therefore, if p has some derivation, then there then there is execution of M that will accept X_{init}.

Conversely, suppose M accepts its input. A simple trace of the machine's execution will produce a symbolic-derivation of $p(X, W)$ which must contain R1. ∎

To show the claim for $SI(r, q, \Sigma_{\mathcal{P}}, DI_2, M1)$, we modify $\mathcal{P}$ as follows. We replace the rule R3 by

R3′: $e(X, W) \Rightarrow a(X, W)$

and we add the rule

R4: $a(W, X) \Rightarrow a(X, W)$.

The reduction follows from the following theorem:

Theorem A.5: *$SI(\mathtt{R4}, p, \Sigma_{\mathcal{P}}, DI_2, M1)$ holds if and only if M does not accept its input with the initial state X_{init}.*

Proof: The proof is similar to that of Theorem A.4, with the following differences. In any minimal derivation of the query that uses R4 the variables X and W must be distinct. Moreover, every derivation of the query that does not use R4 can be modified to use R4. We simply apply R4 to a subgoal of rule R2 and R3 to the subgoal of R4.

[5]This assumes each block is at least of size 3. If this doesn't hold, we simply add another dummy column to each block, and leave it unchanged in all the rules.

Therefore, if there is a minimal derivation of p that includes R4, then M will accept its input.

Conversely, suppose M accepts its input. We can assume the machine does not enter a state with an id identical to one of its ancestors. A simple trace of the machine's execution will produce a minimal symbolic-derivation of $p(X, W)$, and like before, it can be made to contain R4. ∎

Finally, it should be noted that the size of the program $\mathcal{P}$ is linear in n, the size of the input tape of M, therefore the construction takes time linear in the size of the input.

A.2 Proofs of Chapter 4

Proof of Theorem 4.6

In proving the theorem we will use the following lemma:

Lemma A.6: *Let P and Q be two datalog programs such that $P \supseteq Q$, i.e., for any given database D, the answers obtained for the query predicate of P is a superset of those obtained for the query predicate of Q. The problem of determining whether P and Q are equivalent (i.e., produce the same answer for any database D) is undecidable.*

Proof: In [Shmueli, 1987] it is shown that determining whether two arbitrary datalog programs are equivalent is undecidable. Suppose there is an algorithm A to determine equivalence of two programs P and Q when it is known that $P \supseteq Q$. Let S and R be two arbitrary datalog programs, and assume (without loss of generality) that they do not share any IDB predicates. Let G be the program consisting of the rules of S and R and the following rules:

$s(\bar{X}) \Rightarrow g(\bar{X})$
$r(\bar{X}) \Rightarrow g(\bar{X})$

where g is the query predicate of G. The result of G is the union of the results of S and R, and therefore, $G \supseteq S$ and $G \supseteq R$. Clearly, the programs S and R are equivalent if and only if

- G and S are equivalent and
- G and R are equivalent.

Each of these can be determined by the algorithm A. Therefore, the existence of A leads to a contradiction. ∎

Based on this lemma we can prove Theorem 4.6 as follows.
Proof: We will show that if determining $SI(\phi_1, q, \Sigma', DI_2, \mathcal{D}_q)$ is decidable, this will contradict Lemma A.6. Let P and Q be two datalog programs with query predicates p and q respectively and such that $P \supseteq Q$, and assume without loss of generality that P and Q have no IDB predicates in common. Consider the following program G that includes the rules of P and Q and the following rules for its query predicate g:

$r_1 : p(\bar{X}) \wedge e(\bar{X}) \Rightarrow g(\bar{X})$.
$r_2 : q(\bar{X}) \wedge e(\bar{X}) \Rightarrow g(\bar{X})$.

where e is a new EDB predicate that appears nowhere in P or Q and has the same arity as p and q.

To prove the theorem we establish the following claim. Let I be the irrelevance claim that states that r_2 is strongly irrelevant to g. Then $I \Rightarrow SI(r_1, g, \Sigma', DI_2, \mathcal{D}_q)$ if and only if P and Q are equivalent.

If P and Q are equivalent, then the join of q and e is empty exactly when the join of p and e is empty. Therefore, if D is a database in which r_2 is not used in any derivation of g, then r_1 will not be used in any derivation of g either.

Suppose P and Q are not equivalent, i.e., $P \supset Q$, and $P \neq Q$. Then there is some database D in which the difference between p and q (denoted by $p - q$) is not-empty. Consider the database D', that consists of D and the facts $e(\bar{X})$ for $\bar{X}$ such that $\bar{X} \in p - q$. The database D' is such that r_2 will not be used in any derivation of g, however, r_1 will be used. Therefore, $I \Rightarrow SI(r_1, g, \Sigma', DI_2, \mathcal{D}_q)$ cannot hold. ∎

A.3 Proofs of Chapter 6

In our proofs we use the following lemma that is proven by Plaisted [Plaisted, 1981]:

Lemma A.7: *Let f be a mapping on literals, which is extended in a straightforward fashion to a mapping on clauses. Suppose f satisfies the following properties:*

1. *$f(\neg L) = \neg f(L)$ for any literal L.*
2. *If C and D are clauses and D is an instance of C, then $f(D)$ is an instance of $f(C)$.*

Let C_1 and C_2 be two clauses and $C3$ be one of their resolvents. Then the clause $f(C_3)$ is subsumed by some resolvent of $f(C_1)$ and $f(C_2)$.

This lemma implies the following proposition:

Proposition A.8:

1. *Let C_1 and C_2 be clauses that are independent of the predicate arguments $\mathcal{R}$, and suppose C_3 is a resolvent of C_1 and C_2. Then $f_{\mathcal{R}}(C_3)$ is subsumed by some resolvent of $f_{\mathcal{R}}(C_1)$ and $f_{\mathcal{R}}(C_2)$.*

2. *Let C_1 and C_2 be clauses that are independent of the predicate refinement $\mathcal{Q}$, and suppose C_3 is a resolvent of C_1 and C_2. Then $f_{\mathcal{Q}}(C_3)$ is subsumed by some resolvent of $f_{\mathcal{Q}}(C_1)$ and $f_{\mathcal{Q}}(C_2)$.*

Proof: The proof follows from the observation that both mappings, $f_{\mathcal{R}}$ and $f_{\mathcal{Q}}$ satisfy the conditions of Lemma A.7. ∎

Proof of Theorem 6.5

Proof: The first half of the theorem follows from Proposition A.8. Let D be the derivation for which $DI(\mathcal{R}, D)$ holds. By the proposition, if $Base(D) \vdash C$ then there is some clause C' that subsumes $f_{\mathcal{R}}(C)$ such that $f_{\mathcal{R}}(Base(D)) \vdash C'$. Therefore, $f_{\mathcal{R}}(Base(D)) \vdash f_{\mathcal{R}}(C)$.

For the converse, suppose $f_{\mathcal{R}}(\Delta) \models f_{\mathcal{R}}(q)$ and let I be a model of Δ, i.e., $I \models \Delta$. We need to show that $I \models q$. By the definition of independence, $Abs(I) \models Abs_{\mathcal{R}}(\Delta)$ and therefore, $Abs(I) \models f_{\mathcal{R}}(q)$. However, since q contains no irrelevant arguments, the relations denoted by predicates occurring in q and identical to the relations denoted by predicates occurring in $f_{\mathcal{R}}(q)$, and therefore $I \models q$. ∎

Proof of Theorem 6.7

Proof: Suppose that A1–A3 hold as required. Let I be a model of Δ (and therefore, $I \models C$), and let μ' be an arbitrary variable assignment to the variables of $f_{\mathcal{R}}(C)$. To show that $Abs(I) \models f_{\mathcal{R}}(C)$ we need to show that $Abs(I) \models f_{\mathcal{R}}(C)\mu'$.

Note that if μ is an arbitrary extension of μ' to the variables of C, then since $I \models C$ holds, then $I \models C\mu$ holds. Therefore, I satisfies at least one of the literals of $C\mu$. There are three possible cases:

Case 1: There is some extension μ, such that one of the satisfied literals L is of a predicate p (either positive or negative) and p does not appear in $\mathcal{R}$. In this case, $f_{\mathcal{R}}(L) = L$ holds and therefore $Abs(I) \models f_{\mathcal{R}}(L)$ holds because the relation denoted by p in I is the same as denoted in $Abs(I)$.

Case 2: There is some extension μ, such that one of the satisfied literals is a positive literal $p(\bar{X})$, where p appears in $\mathcal{R}$, and $f_{\mathcal{R}}(p(\bar{X})) = p'(\bar{Y})$. If $I \models p(\bar{X})\mu$ then

$Abs(I) \models p'(\bar{Y})\mu'$ since $\bar{Y}\mu'$ is a projection of $X\mu$ and the relation denoted by p' is the corresponding projection of the relation denoted by p.

Case 3: If neither of the first two cases happen then it must be the case that for every extension μ of μ' the literal in C that is satisfied is of the form $\neg p(\bar{X})$ where p appears in $\mathcal{R}$. Again, we denote $f_{\mathcal{R}}(\neg p(\bar{X}))$ by $\neg p'(\bar{Y})$. In order to prove that C' is independent, we need to show that there is a *single* literal $\neg p_i(\bar{X}_i) \in Neg(C)$ such that for *any* extension μ of μ', $I \models \neg p_i(\bar{X}_i)\mu$. If there exists such a literal, then $Abs(I) \models \neg p'_i(\bar{Y}_i)$ holds because the projection of the relation denoted by p_i on $\bar{Y}_i\mu'$ is empty. This follows from the fact that no constants appear in argument positions of $\neg p_i(\bar{X}_i)$ that are projected.

To prove that there exists such a literal, assume the contrary. That means we assume that for every negative literal $\neg p_i(\bar{X}_i) \in C$ such that p_i appears in $\mathcal{R}$, there is some extension μ of μ' such that $I \models p_i(\bar{X}_i)\mu$. To show the contradiction, we will build an extension μ_0 of μ' such that $I \not\models C\mu_0$.

If $(p, i) \in \mathcal{R}$ and X is a variable such that $X \in AtPos(p, i)$, then X does not appear in $f_{\mathcal{R}}(C)$. This is because it has only one appearance in $Neg(C)$ (by A2) and all its appearances in $Pos(C)$ have been projected out (by A3). Therefore, in extending μ' to μ_0 we are free to assign a value to X. If $\neg p_i(\bar{X}_i) \in Neg(C)$, then in μ_0 we assign to the variables in $\bar{X}_i - \bar{Y}_i$[6] the values that make $p_i(\bar{X}_i)$ satisfiable in I. Note that such an assignment exists because of our assumption. Furthermore, the choice of assignments for the variables in $\bar{X}_i - \bar{Y}_i$ does not affect variables in the other literals of $Neg(C)$ (because of A2), and therefore can be done independently for every literal in $Neg(C)$. Variables in $Var(C) - Var(f_{\mathcal{R}}(C))$ that appear only in $Pos(C)$ are assigned arbitrary values. Because of our assumptions, none of the literals in $Neg(C)$ are satisfied with the variable assignment μ_0. Moreover, none of the positive literals are satisfied because neither cases 1 or 2 occurred. Therefore, $I \not\models C$ holds which is a contradiction. ∎

Proof of Theorem 6.9

Proof: The first half of the theorem follows from Proposition A.8. Let D be the derivation for which $DI(\mathcal{Q}, D)$ holds. By the proposition, if $Base(D) \vdash \phi$ then there is some clause ϕ' that subsumes $f_{\mathcal{Q}}(\phi)$ such that $f_{\mathcal{Q}}(Base(D)) \vdash \phi'$. Therefore, $f_{\mathcal{Q}}(Base(D)) \vdash f_{\mathcal{Q}}(\phi)$.

For the converse, suppose $f_{\mathcal{Q}}(\Delta) \models f_{\mathcal{Q}}(q)$ and let I be a model of Δ, i.e., $I \models \Delta$. We need to show that $I \models q$. By the definition of independence $Abs(I) \models Abs_{\mathcal{Q}}(\Delta)$ holds and therefore, $Abs(I) \models f_{\mathcal{Q}}(q)$. However, since q does not contain predicates from $\mathcal{Q}$, the relations denoted by predicates occurring in q are identical to the relations

[6] $\bar{X}_i - \bar{Y}_i$ denotes the variables that appear in $\bar{X}_i$ and not in $\bar{Y}_i$

denoted by predicates occurring in $f_{\mathcal{Q}}(\dot{q})$, and therefore $I \models q$. ∎

Proof of Lemma 6.10

Proof: Suppose the clause C is independent of the predicate refinement $\mathcal{Q}$. Let $Neg(C)'$ be the result of replacing occurrences of predicates in $\mathcal{Q}$ by arbitrary other predicates in $\mathcal{Q}$. We define C_2 as follows. C_2 includes the literals in $Pos(C)$ as well as the following literals. If $p(\bar{X}) \in Pos(C)$ and $p \in \mathcal{Q}$, then C_2 includes the literals $q_1(\bar{X}), \ldots, q_n(\bar{X})$. We denote by C' the clause containing the union of literals in $Neg(C)'$ and C_2. Note that $f_{\mathcal{Q}}(C_2) = f_{\mathcal{Q}}(Pos(C))$. We show that $\Delta \models C'$.

Let I be a model of Δ (and therefore of C) and let μ be an arbitrary assignment to the variables in C (which are the same as the variables in C'). We need to show that $I \models C'\mu$.

Clearly, $I \models C\mu$ and so some literal in $C\mu$ is satisfied by I. If the satisfied literal is either positive or involves a predicate that does not appear in $\mathcal{Q}$, then the same literal will appear in C' and therefore, $I \models C'\mu$.

Otherwise, the satisfied literal is of the form $\neg q_i(\bar{X})$, where $q_i \in \mathcal{Q}$. In C' the literal $\neg q_i(\bar{X})$ is mapped to $\neg q_j(\bar{X})$. Recall that by our assumption, $Abs(I) \models f_{\mathcal{Q}}(C)$, and therefore, $Abs(I) \models f_{\mathcal{Q}}(C)\mu$. Let $L = r(\bar{Y})$ be a literal (either positive or negative) satisfied in $f_{\mathcal{Q}}(C)\mu$. There are three cases:

Case 1: There is a satisfied literal such that $r \notin \mathcal{Q}$. In this case, the literal $r(\bar{Y})$ appears in C and C' and therefore $I \models C'\mu$.

Case 2: There is a satisfied positive literal L of the form $\dot{q}(\bar{X})$ where q is the new predicate. This means that for some $q_i \in \mathcal{Q}$, $q_i(\bar{X})\mu \in Q_i$, where Q_i is the relation denoted by q_i in I. The literal $\dot{q}_i(\bar{X})$ is also in C_2, and therefore $I \models C'\mu$.

Case 3: The satisfied literal L is negative and of the form $\neg\dot{q}(\bar{X})$. This means that for all predicates $\dot{q}_i \in \dot{\mathcal{Q}}$, $q_i(\bar{X})\mu \notin Q_i$. In particular, it is true for $\dot{q}_j$, and therefore, the literal $\neg q_j(\bar{X})\mu$ is satisfied, and $I \models C'\mu$.

For the other direction, let C be a clause and assume the condition of the lemma holds. Let I be a model of Δ. We need to show that $Abs(I) \models f_{\mathcal{Q}}(C)$.

Let μ be an arbitrary assignment to the variables of C. We need to show that $Abs(I) \models f_{\mathcal{Q}}(C)\mu$. Clearly, $I \models C\mu$. There are three cases:

Case 1: If one of the satisfied literals involves a predicate that is not in $\mathcal{Q}$, then that literal will also appear identically in $f_{\mathcal{Q}}(C)$ and the relation denoted by the predicate of the literal will be the same in I and $Abs(I)$. Therefore, $Abs(I) \models f_{\mathcal{Q}}(C)\mu$.

Case 2: If one of the satisfied literals is a positive literal of the form $\dot{q}_i(\bar{X})$, where $q_i \in \mathcal{Q}$, then corresponding literal in $f_{\mathcal{Q}}(C)$ will be $q(\bar{X})$. However, since the relation denoted by $\dot{q}$ in $Abs(I)$ is includes the relation denoted by q_i in I, then $Abs(I)$ will

satisfy $q(\bar{X})\mu$. Moreover, if there is any literal of the form $q_i(\bar{X})$, where $q_i \in \mathcal{Q}$ such that $I \models q_i(\bar{X})\mu$, then $Abs(I) \models q(\bar{X})\mu$.

Case 3: If neither of the previous cases occurred, then the set of satisfied literals must be of the form $\neg q_1(\bar{X}_1), \ldots, \neg q_k(\bar{X}_k)$, where $q_i \in \mathcal{Q}$ for $1 \leq i \leq k$. To complete the proof by showing that for at least one of these literals, $Abs(I) \models \neg q(\bar{X}_i)$. Suppose the contrary, i.e., for each of these literals $Abs(I) \not\models q(\bar{X}_i)$. This means that for i, $1 \leq i \leq k$ there exists a predicate $q_{g(i)}$, such that $q_{g(i)} \in \mathcal{Q}$ and $\bar{X}_i\mu \in Q_{g(i)}$. Consider the set of literals $Neg(C)'$ obtained by replacing $q_i(\bar{X}_i)$ by $q_{g(i)}(\bar{X}_i)$. By the assumption of the lemma, there exists some set of literals C_2 such that $f_{\mathcal{Q}}(C_2) = f_{\mathcal{Q}}(Pos(C))$ and $\Delta \models Neg(C)' \cup C_2$ and therefore, $I \models (Neg(C)' \cup C_2)\mu$. Clearly, each of the satisfied literals in $(Neg(C)' \cup C_2)\mu$ must be a positive literal involving a predicate in $\mathcal{Q}$. However, since $f_{\mathcal{Q}}(C_2) = f_{\mathcal{Q}}(Pos(C))$, this would contradict the fact that case 2 did not occur. ∎

Bibliography

[Abiteboul and Hull, 1988] Abiteboul, Serge and Hull, Richard 1988. Data functions, datalog and negation. In *Proceedings of ACM SIGMOD 1988 International Conference on Management of Data.*

[Addanki *et al.*, 1989] Addanki, Sanjaya; Cremonini, R.; and Penberthy, J. 1989. Reasoning about assumptions in graphs of models. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.*

[Alchourron *et al.*, 1985] Alchourron, C.; Gärdenfors, Peter; and Makinson, D. 1985. On the logic of theory change: Partial meet contraction and revision functions. *Journal of Symbolic Logic* 50:510–530.

[Amarel, 1981] Amarel, Saul 1981. On representations of problems of reasoning about actions. In Webber, Bonnie L. and Nilsson, Nils J., editors 1981, *Readings in Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA.

[Anderson and Belnap, 1975] Anderson, Alan R. and Belnap, Nuel D. 1975. *Entailment.* Princeton University Press, Princeton, New Jersey.

[Avron, 1992] Avron, Arnon 1992. Whither relevance logic? *Journal of Philosophical Logic* 21:243–281.

[Bacchus and Yang, 1992] Bacchus, Fahiem and Yang, Qiang 1992. The expected value of hierarchical problem-solving. In *Proceedings of the Tenth National Conference on Artificial Intelligence.* 369–374.

[Bacchus *et al.*, 1993] Bacchus, F.; Grove, A. J.; Halpern, J. Y.; and Koller, D. 1993. Statistical foundations for default reasoning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence.*

[Blakeley *et al.*, 1989] Blakeley, J. A.; Coburn, N.; and Larson, P. A. 1989. Updating derived relations: detecting irrelevant and autonomously computable updates. *Transactions of Database Systems* 14(3):369–400.

[Bruynooghe *et al.*, 1989] Bruynooghe, Maurice; De-Schreye, Danny; and Krekels, B. 1989. Compiling control. *Journal of Logic Programming* (6) 135-162.

[Bruynooghe *et al.*, 1991] Bruynooghe, Maurice; De-Schreye, Danny; and Martens, Bern 1991. A general criterion for avoiding infinite unfolding during partial deduction of logic programs. In *Proceedings of the International Symposium on Logic Programming*. 117-131.

[Carnap, 1950] Carnap, R. 1950. *Logical Foundations of Probability*. University of Chicago Press, Chicago.

[Chandra *et al.*, 1981] Chandra, Ashok; Kozen, Dexter; and Stockmeyer, Larry 1981. Alternation. *Journal of the ACM* 28(1): 114-133.

[Chang, 1979] Chang, C. L. 1979. Resolution plans in theorem proving. In *Proceedings of the Sixth International Joint Conference on Artificial Intelligence*. 143-148.

[Clancey, 1983] Clancey, William J. 1983. The advantages of abstract control knowledge in expert system design. In *Proceedings of the Third National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann. 74-78.

[Cutkosky *et al.*, 1993] Cutkosky, Mark R.; Engelmore, Robert S.; Fikes, Richard E.; Genesereth, Michael R.; Gruber, Thomas R.; Mark, William S.; Tenenbaum, Jay M.; and Weber, Jay C. 1993. PACT: An experiment in integrating concurrent engineering systems. *IEEE Computer* 26(1):28-37.

[Dalal, 1988] Dalal, Mukesh 1988. Investigations into the theory of knowledge base revision: preliminary report. In *Proceedings of the Seventh National Conference on Artificial Intelligence*. 475-479.

[de Kleer and Brown, 1984] de Kleer, J. and Brown, J. S. 1984. A qualitative physics based on confluences. *Artificial Intelligence* 24:7-83.

[de Kleer and Brown, 1986] de Kleer, J. and Brown, J. S. 1986. Theories of causal ordering. *Artificial Intelligence* 29:33-61.

[de Kleer, 1986] de Kleer, J. 1986. An assumption-based TMS. *Artificial Intelligence* 28:127-162.

[Dunn, 1986] Dunn, Michael 1986. Relevance logic and entailment. In Gabbay, Dov and Guenthner, Franz, editors 1986. *Handbook of Philosophical Logic, Volume III. Alternatives to Classical Logic*. D. Reidel Publishing Company, Dordrecht, Holland. 117-224.

[Elkan, 1990] Elkan, Charles 1990. Independence of logic database queries and updates. In *Proceedings of the 9th ACM Symp. on Principles of Database Systems.* 154–160.

[Ellman, 1990] Ellman, Thomas, editor 1990. *Working Notes of the Automatic Generation of Approximations and Abstractions Workshop.* American Association for Artificial Intelligence.

[Ellman, 1992] Ellman, Thomas, editor 1992. *Working Notes of the Workshop on Approximation and Abstraction of Computational Theories.* American Association for Artificial Intelligence.

[Enderton, 1972] Enderton, Herbert B. 1972. *A Mathematical Introduction to Logic.* Academic Press, Inc., Orlando, Florida.

[Etherington *et al.*, 1989] Etherington, David; Borgida, Alex; Brachman, Ronald J.; and Kautz, Henry-1989. Vivid knowledge and tractable reasoning: Preliminary report. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Los Altos, CA. Morgan Kaufmann. 1146–1152.

[Etzioni and Minton, 1992] Etzioni, Oren and Minton, Steven 1992. Why EBL produces overly-specific knowledge: A critique of the PRODIGY approaches. In *Proceedings of the Machine Learning Conference.*

[Etzioni, 1990] Etzioni, Oren 1990. Why PRODIGY/EBL works. In *Proceedings of the Eighth National Conference on Artificial Intelligence.*

[Etzioni, 1993] Etzioni, Oren 1993. Acquiring search-control knowledge via static analysis. *Artificial Intelligence* 62.

[Fagin *et al.*, 1983] Fagin, Ronald; Ullman, Jeffrey; and Vardi, Moshe 1983. On the semantics of updates in databases. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* 352–365.

[Falkenhainer and Forbus, 1991] Falkenhainer, Brian and Forbus, Ken 1991. Compositional modeling: Finding the right model for the job. *Artificial Intelligence* 51:95–143.

[Farquhar *et al.*, 1993] Farquhar, Adam; Bobrow, Danny; Falkenhainer, Brian; Fikes, Richard; Forbus, Kenneth; Gruber, Thomas; Iwasaki, Yumi; and Kuipers, Benjamin 1993. A compositional modeling language. Knowledge Systems Laboratory Technical Report KSL-93-53. Stanford University, Stanford, California.

[Fikes *et al.*, 1991] Fikes, Richard; Cutkosky, Mark; Gruber, Thomas; and Van Baalen, Jeffrey 1991. Knowledge sharing technology, project overview. Knowledge Systems Laboratory technical report No. KSL 91-71.

[Forbus, 1984] Forbus, Kenneth D. 1984. Qualitative process theory. *Artificial Intelligence* 24:85-168.

[Gärdenfors, 1978] Gärdenfors, Peter 1978. On the logic of relevance. *Sythese* 37:351-367.

[Gärdenfors, 1988] Gärdenfors, Peter 1988. *Knowledge in Flux: Modeling the Dynamics of Epistemic States.* MIT Press, Cambridge, MA.

[Geffner and Pearl, 1990] Geffner, Hector and Pearl, Judea 1990. A framework for reasoning with defaults. In Kyburg, H.E.; Loui, R.; and Carlson, G, editors 1990, *Knowledge Representation and Defeasible Reasoning.* Academic Press, Dordrecht, Netherlands.

[Genesereth and Nilsson, 1987] Genesereth, Michael R. and Nilsson, Nils J. 1987. *Logical Foundations of Artificial Intelligence.* Morgan Kaufmann, Los Altos, CA.

[Genesereth, 1992] Genesereth, Michael R. 1992. An agent-based framework for software interoperability. In *Proceedings of the Software Technology Conference, Los Angeles, CA.*

[Giunchiglia and Walsh, 1992] Giunchiglia, Fausto and Walsh, Toby 1992. A theory of abstraction. *Artificial Intelligence* 56 (3).

[Graham *et al.*, 1989] Graham, Ronald L.; Knuth, Donald E.; and Patashnik, Oren 1989. *Concrete Mathematics: A Foundation for Computer Science.* Addison Wesley.

[Greiner and Jurišica, 1992] Greiner, Russell and Jurišica, Igor 1992. A statistical approach to solving the EBL utility problem. In *Proceedings of the Tenth National Conference on Artificial Intelligence.*

[Greiner, 1980] Greiner, Russell 1980. RLL-1: A representation language language. Stanford Heuristic Programming Project, HPP-80-9 (Working Paper).

[Greiner, 1991] Greiner, Russell 1991. Finding optimal derivation strategies in a redundant knowledge base. *Artificial Intelligence* 50(1):95-116.

[Greiner, 1992] Greiner, Russell 1992. Learning efficient query processing strategies. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*

[Guha, 1991] Guha, Ramanathan V. 1991. *Contexts: A Formalization and Some Applications.* Ph.D. Dissertation, Stanford University, Stanford, CA.

[Hayes, 1973] Hayes, Patrick J. 1973. Computation and deduction. In *Proceedings of the 1973 Mathematical Foundations of Computer Science Symposium. Czechoslovakian Academy of Sciences.*

[Hobbs, 1985] Hobbs, Jerry R. 1985. Granularity. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence,* Los Altos, CA. Morgan Kaufmann. 432-435.

[Imielinski, 1987] Imielinski, Tomasz 1987. Domain abstraction and limited reasoning. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Los Altos, CA. Morgan Kaufmann. 997-1003.

[Iwasaki and Levy, 1993] Iwasaki, Yumi and Levy, Alon Y. 1993. Automated model selection for simulation. In *Proceedings the Seventh International Workshop on Qualitative Reasoning about Physical Systems.*

[Iwasaki and Low, 1992] Iwasaki, Yumi and Low, Chee Meng 1992. Device modeling environment: An integrated model-formulation and simulation environment for continuous and discrete phenomena. In *Proceedings of Conference on Intelligent Systems Engineering.*

[Iwasaki and Simon, 1986] Iwasaki, Yumi and Simon, Herb 1986. Causality in device behavior. *Artificial Intelligence* 29:3-32.

[Kanellakis *et al.*, 1990] Kanellakis, P.C.; Kuper, G.M.; and Revesz, P.Z. 1990. Constraint query languages. In *Proceedings of the 9th ACM Symp. on Principles of Database Systems.* 299-313.

[Keynes, 1921] Keynes, J. M. 1921. *A Treatise on Probability.* Macmillan, London.

[Kifer, 1988] Kifer, M. 1988. On safety, domain independence, and capturability of database queries. In *Proceedings of the International Conference on Data and Knowledge Bases, Jerusalem.*

[Klug, 1988] Klug, A. 1988. On conjunctive queries containing inequalities. *Journal of the ACM* 35(1): 146-160.

[Knoblock *et al.*, 1991] Knoblock, Craig; Tenenberg, Josh D.; and Yang, Qiang 1991. Characterizing abstraction hierarchies for planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence,* Cambridge MA. MIT Press. 692-697.

[Knoblock, 1990] Knoblock, Craig A. 1990. Learning abstraction hierarchies for problem solving. In *Proceedings of the Eighth National Conference on Artificial Intelligence*. Los Altos, CA. Morgan Kaufmann.

[Kowalski, 1975] Kowalski, Robert 1975. A proof procedure using connection graphs. *Journal of the ACM* 22(4): 572-595.

[Levesque, 1989] Levesque, Hector 1989. A knowledge-level account of abduction. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*.

[Levitt *et al.*, 1991] Levitt, Raymond E.; Jin, Yan; and Dym, Clive L. 1991. Knowledge based support for management of concurrent, multidisciplinary design. *AI in Engineering, Design and Manufacturing* 2(5):77-95.

[Levy and Sagiv, 1992] Levy, Alon Y. and Sagiv, Yehoshua 1992. Constraints and redundancy in datalog. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*

[Levy and Sagiv, 1993a] Levy, Alon Y. and Sagiv, Yehoshua 1993a. Exploiting irrelevance reasoning to guide problem solving. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*.

[Levy and Sagiv, 1993b] Levy, Alon Y. and Sagiv, Yehoshua 1993b. Queries independent of updates. In *Proceedings of the 19th VLDB Conference, Dublin, Ireland.*

[Levy *et al.*, 1992] Levy, Alon; Iwasaki, Yumi; and Motoda, Hiroshi 1992. Using relevance reasoning to guide compositional modeling. In *The Proceedings of the Second Pacific Rim International Conference on Artificial Intelligence, Seoul, South Korea. Also appears in the Proceedings of the Workshop on Approximations and Abstractions of Computational Theories, AAAI-92, San Jose, CA.*

[Levy *et al.*, 1993] Levy, Alon Y.; Mumick, Inderpal Singh; Sagiv, Yehoshua; and Shmueli, Oded 1993. Equivalence, query-reachability and satisfiability in datalog extensions. In *Proceedings of the Twelfth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Washington D.C.*

[Litwin *et al.*, 1990] Litwin, Witold; Mark, Leo; and Roussopoulos, Nick 1990. Interoperability of multiple autonomous databases. *ACM Computing Surveys* 22 (3):267-293.

[Lloyd and Shepherdson, 1991] Lloyd, J.W. and Shepherdson, J.C. 1991. Partial evaluation in logic programming. *Journal of Logic Programming* (11):217-242.

[Lowry, 1992] Lowry, Michael R., editor 1992. *Proceedings of the Workshop on Change of Representation and Problem Reformulation.* NASA Ames Research Center Technical Report FIA-92-06.

[Minton *et al.*, 1989] Minton, Steven; Carbonell, Jaime; Knoblock, Craig; Kuokka, D.; Etzioni, Oren; and Gil, Yolanda 1989. Explanation based learning: A problem solving perspective. *Artificial Intelligence* 40:63–118.

[Minton, 1988] Minton, Steve 1988. Quantitative results concerning the utility of explanation based learning. In *Proceedings of the Seventh National Conference on Artificial Intelligence.*

[Mumick *et al.*, 1990] Mumick, I. S.; Finkelstein, S.J.; Pirahesh, H.; and Ramakrishnan, R. 1990. Magic conditions. In *Proceedings of the 9th ACM Symposium on Principles of Database Systems.* 314–330.

[Nayak, 1992a] Nayak, Pandurang 1992a. *Automated Model Selection.* Ph.D. Dissertation, Stanford University, Stanford, CA.

[Nayak, 1992b] Nayak, Pandurang 1992b. Causal approximations. In *Proceedings of the Tenth National Conference on Artificial Intelligence.*

[Nebel, 1989] Nebel, B. 1989. A knowledge level analysis of belief revision. In *Proceedings of KR-89.* 301–311.

[Pearl, 1988] Pearl, Judea 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

[Pearl, 1990] Pearl, Judea 1990. System Z: A natural ordering of defaults with tractable applications to nonmonotic reasoning. In Vardi, Moshe Y., editor 1990, *Theoretical Aspects of Reasoning About Knowledge.* Morgan Kaufmann Publishers. 121–135.

[Plaisted, 1981] Plaisted, D. 1981. Theorem proving with abstraction. *Artificial Intelligence* 16:47–108.

[Ramakrishnan *et al.*, 1988] Ramakrishnan, Raghu; Beeri, Catriel; and Krishnamurthy, Ravi 1988. Optimizing existential datalog queries. In *Proceedings of the Seventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, Austin TX.* 89–101.

[Rickel and Porter, 1992]. Rickel, Jeff and Porter, Bruce 1992. Automated modeling for answering prediction questions: Exploiting interaction paths. In *Proceedings of the Sixth International Workshop on Qualitative Reasoning about Physical Systems.*

[Rombauer and Rombauer-Becker, 1975] Rombauer, Irma S. and Rombauer-Becker, Marion 1975. *Joy of Cooking.* Bobbs Merrill Company Inc., N.Y.C., N.Y.

[Russell, 1985] Russell, Stuart 1985. The complete guide to MRS. Technical Report KSL-85-12, Knowledge Systems Laboratory, Department of Computer Science, Stanford University, CA.

[Sacerdoti, 1974] Sacerdoti, Earl D. 1974. Planning in a hierarchy of abstraction spaces. *Artificial Intelligence* 5:115-135.

[Sagiv, 1988] Sagiv, Yehoshua 1988. Optimizing datalog progra... In Minker, Jack, editor 1988, *Foundations of Deductive Databases and Logic Programming.* Morgan Kaufmann, Los Altos, CA. 659-698.

[Selman and Kautz, 1991] Selman, Bart and Kautz, Henry 1991. Knowledge compilation using horn approximations. In *Proceedings of the Ninth National Conference on Artificial Intelligence,* Cambridge MA. MIT Press. 904-909.

[Shmueli, 1987] Shmueli, Oded 1987. Decidability and expressiveness aspects of logic queries. In *Proceedings of the 6th ACM Symposium on Principles of Database Systems.* 237-249.

[Sickel, 1976] Sickel, Susan 1976. A search technique for clause interconnectivity graphs. *IEEE Transactions on Computers* C-25(8):823-835.

[Slutzki, 1985] Slutzki, G. 1985. Alternating tree automata. *Theoretical Computer Science* 41:305-318.

[Smith and Hickey, 1990] Smith, Donald A. and Hickey, Timothy J. 1990. Partial evaluation of a CLP language. In *Proceedings of the International Symposium on Logic Programming.* 119-138.

[Smith, 1986] Smith, David 1986. *Controlling Inference.* Ph.D. Dissertation, Stanford University, Stanford, CA.

[Srivastava and Ramakrishnan, 1992] Srivastava, Divesh and Ramakrishnan, Raghu 1992. Pushing constraint selections. In *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.*

[Subramanian and Genesereth, 1987] Subramanian, D. and Genesereth, M.R. 1987. The relevance of irrelevance. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence,* Los Altos, CA. Morgan Kaufmann.

[Subramanian, 1989] Subramanian, Devika 1989. *A Theory of Justified Reformulations.* Ph.D. Dissertation, Stanford University, Stanford, CA.

[Tenenberg, 1990] Tenenberg, Josh D. 1990. Abstracting first order theories. In Benjamin, Paul, editor 1990, *Change of Representation and Inductive Bias*. Kluwer, Boston, Mass.

[Ullman, 1989] Ullman, Jeffery D. 1989. *Principles of Database and Knowledge-base Systems, Volumes I, II.* Computer Science Press, Rockville MD.

[van der Meyden, 1992] van der Meyden, Ron 1992. The complexity of querying indefinite data about linearly ordered domains. In *The Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, San Diego, CA.* 331–345.

[Van-Gelder, 1986] Van-Gelder, Allen 1986. A message passing framework for logical query evaluation. In *ACM SIGMOD International Conference on Management of Data.* 155–165.

[Vardi, 1989] Vardi, Moshe Y. 1989. Automata theory for database theoreticians. In *Proceedings of the Eighth Symposium on Principles of Database Systems (PODS).* 83–92.

[Weld, 1990] Weld, Daniel 1990. Approximation reformulation. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Los Altos, CA. Morgan Kaufmann.

[Williams, 1990a] Williams, Brian C. 1990a. Capturing how things work: Constructing critical abstractions of local interactions. In *Proceedings of the AAAI Workshop on Automatic Generation of Approximations and Abstractions, pp. 163-174.*

[Williams, 1990b] Williams, Brian C. 1990b. Interaction-based invention: Designing novel devices from first principles. In *Proceedings of the Eighth National Conference on Artificial Intelligence.*

[Winslett, 1990] Winslett, Marianne 1990. *Updating Logical Databases.* Cambridge University Press, Cambridge, England.